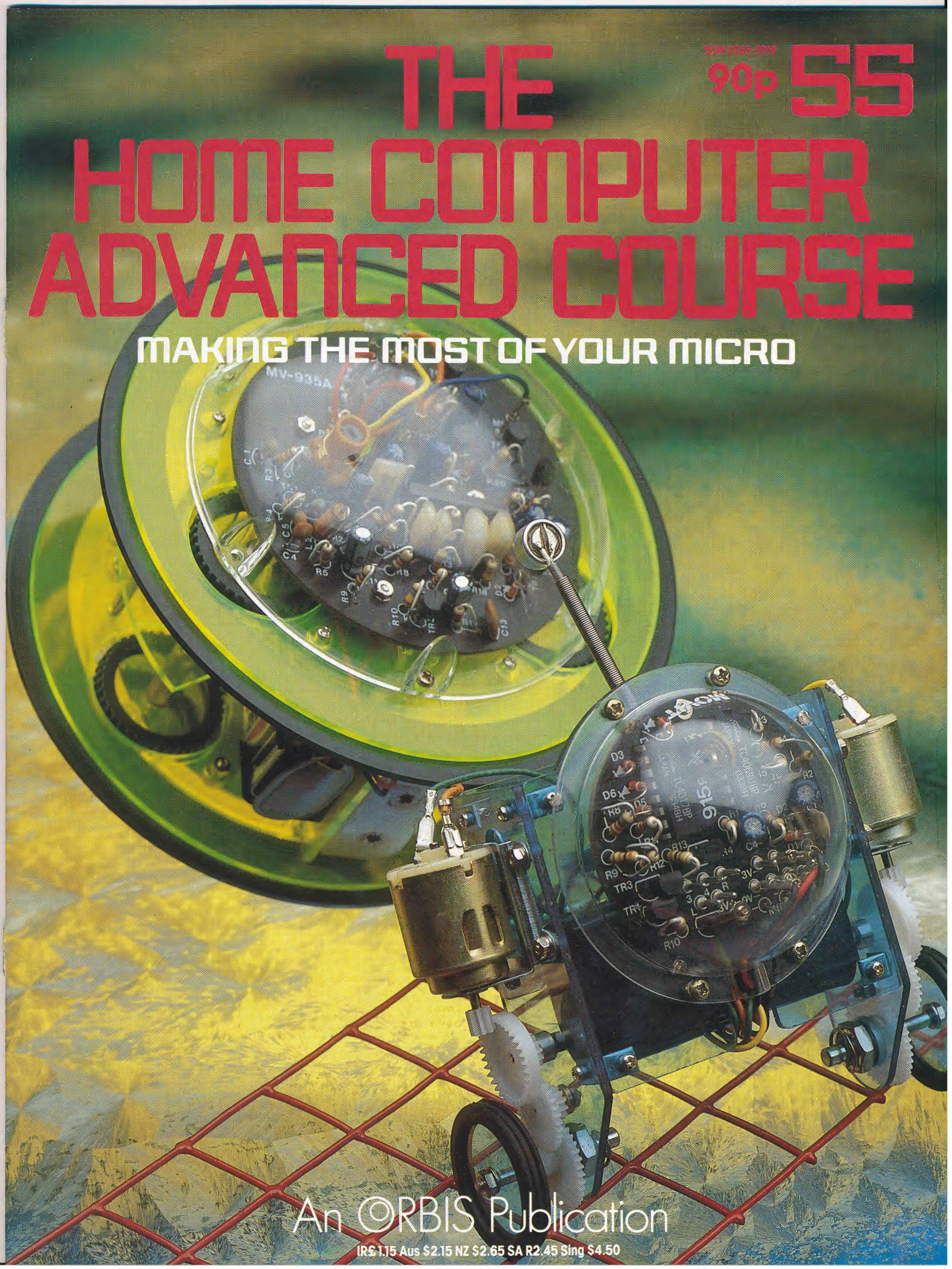
ISSN 0265-2919

90p 55

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

APPLICATION

POWER TO THE PUPIL When children are given the power to control a moving object, they readily grapple with mathematical concepts that they would otherwise find pointless to learn — 1081

HARDWARE

ROBOTS BY NAME Movits are inexpensive and easily-assembled floor 'robots'. We put a few together and contemplate their claim to name — 1084

AMPLE ETUDES The Music 500 synthesiser for the BBC Micro is provided with its own (AMPLE) language — 1090

SOFTWARE

DATA ORGANISATION Databases — what are they, how are they organised, and how are they used? We begin to answer these questions in the first of a new series — 1097

COMPUTER SCIENCE

PLAY IT AGAIN, RAM We compare the three different PASCAL iteration constructs. — 1092

JARGON

FROM MINICOMPUTER TO MODULUS A weekly glossary of computing terms — 1089

PROGRAMMING PROJECTS

LISTED COMMODITIES The final thing we must do, before our ship sets sail, is take on board the goods we are going to trade with in the New World — 1086

MACHINE CODE

THE MAIN EVENTS We examine the most useful of the BBC Micro's 'event' signals — 1099

WORKSHOP

STRONG-ARM TACTICS We provide the cutout patterns for the different parts that make up our robot arm — 1094

REFERENCE CARD The first in a series that will give a comprehensive list of the BBC Micro's OS ROM routine calling addresses — INSIDE BACK COVER

Next Week

- **The Sanyo MBC-550 is a 16-bit computer running MS-DOS for under £1,000. We look at this low-priced business machine.**
- **In our BBC operating system series we look at some practical applications of interrupts and events.**
- **Our PASCAL series continues with an examination of how 'sets' are implemented in the language.**

QUIZ

1) Which of the Movits can be considered a true 'robot' and why?
2) What is the difference between a database and a database manager?
3) In an event handling program for the BBC Micro, where are the start addresses of the event routines stored when they are not being used?
4) What are PASCAL's three iteration constructs?

Answers To Last Week's Quiz

1) Simulations are valuable education aids as they allow children to take the kind of decisions that would have to be made in the real world, providing insights into the reasons for the decisions.
2) It tells the farmer when the cow was last 'served'.
3) *MOUNT places the Currently Selected Directory and Currently Selected Library in memory.
4) The Intel 4004 is generally considered to be the first microprocessor.

Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Production Editor** Bobby Pickering; **Technical Editor** Steve Colwill; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Stephen Malone; **Sub Editor** Jonathan Kaye; **Contributors** Geoff Bains, Nick Walsh, Joe Pritchard, Steve Malone, David Mudd, Anthony Ginn, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Manager** Alastair Gourlay; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd., Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY CHRIS STEVENS

TONY SLEEP

Selling The Idea
Pupils at Southmead Junior School are introduced to the principles of geometry in this exercise using the Valiant turtle. They take an active role in the setting-up of the maze and in programming the turtle to circumnavigate the obstacles. The exercise teaches the principles of angular displacement and other abstract concepts that young children usually find both incomprehensible and unappealing

POWER TO THE PUPIL

A floor robot, or turtle, in the classroom presents the young student with a means of understanding abstract mathematical concepts through a three-dimensional, moving object. As well as being fun to use, robots can provide a new and effective method of teaching.

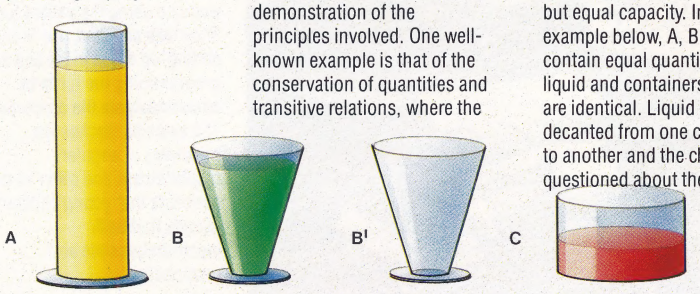
The use of robots in education is an attempt to deal with the problem — especially for young children — of understanding abstract concepts without a concrete model. Complex notions of trigonometry, for example, can be physically demonstrated; mental processes can be represented by the three-dimensional reality of the movements of a turtle. It can also be a useful aid in teaching the fundamentals of computer programming.

In the past, geometry has been taught with instruments such as pencils, rulers and protractors. For an eight year old child, there often seems to be little reason to learn how to measure an angle, let alone attempt to understand the broader concepts of geometry. Angles, on the other hand, take on much more significance when they will determine whether or not a turtle, programmed by the student, will knock over the forest of upended plastic bottles on the classroom floor. Although geometry might remain cloudy at best, there is at least a motivation to understand its practical applications. The difference between 15 and 20cm is soon evident after a choice has been made between a 5° and an 8° angle.

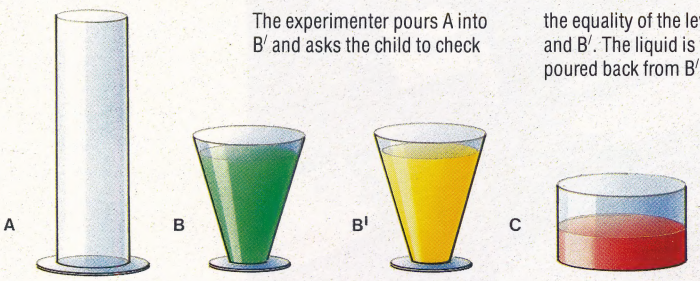
One of the major aspects of turtle technology is that it allows learning by trial and error. A child can modify the programming until a satisfactory course has been plotted, while constantly acquiring insight into the properties of angles, lines and measurements. A mistake means only that something else must be attempted until the desired movements occur — not that something has failed. Programming a robot arm to lift and move an object requires thinking in terms of three planes of movement. Imagining this on paper, which is two-dimensional, is not very easy. With a robot arm, however, the planes of movement are clearly visible. Results of the programming are plainly seen, bugs are easily distinguished and appropriate modifications can be made until it runs correctly. The robot provides instant feedback to the programmer, demonstrating each angle and measurement in sequence far more efficiently than an equivalent exercise using the traditional pencil and paper. This also allows the students to mark their own work.

Concrete Example

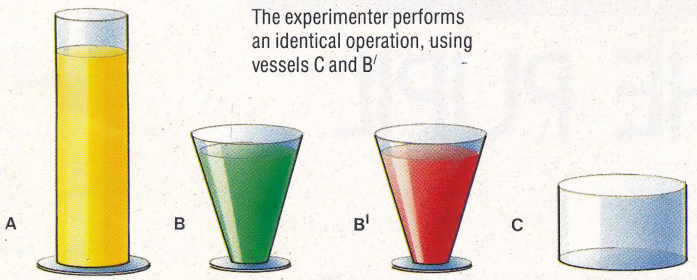
Young children are often incapable of grasping abstract concepts, even when presented with a physical demonstration of the principles involved. One well-known example is that of the conservation of quantities and transitive relations, where the child is presented with containers of different shapes but equal capacity. In the example below, A, B and C contain equal quantities of liquid and containers B and B′ are identical. Liquid is decanted from one container to another and the child is questioned about the result



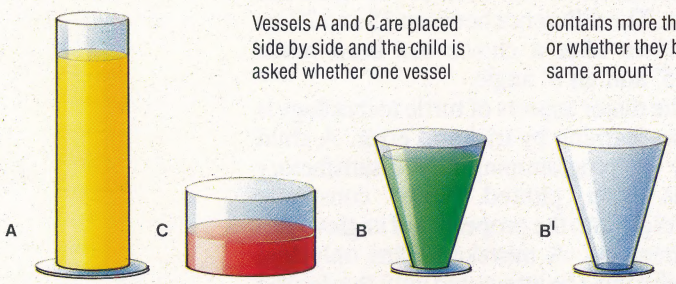
The experimenter pours A into B′ and asks the child to check the equality of the levels in B and B′. The liquid is then poured back from B′ into A



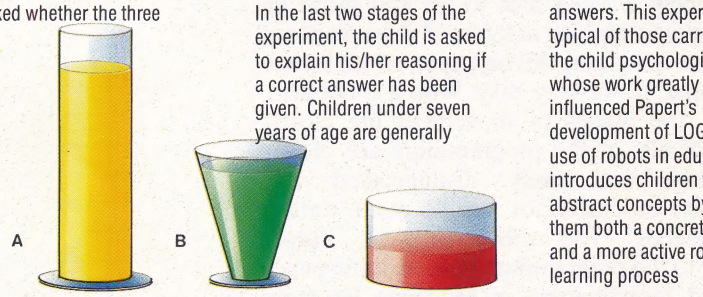
The experimenter performs an identical operation, using vessels C and B′



Vessels A and C are placed side by side and the child is asked whether one vessel contains more than the other, or whether they both hold the same amount



Vessels A, B and C are placed on the table and the child is asked whether the three containers each hold the same amount of water. In the last two stages of the experiment, the child is asked to explain his/her reasoning if a correct answer has been given. Children under seven years of age are generally unable to grasp the principles involved or to give the correct answers. This experiment is typical of those carried out by the child psychologist Piaget, whose work greatly influenced Papert's development of LOGO. The use of robots in education introduces children to abstract concepts by giving them both a concrete model and a more active role in the learning process



KEVIN JONES

A robotic toy currently popular in infant schools (kindergarten to Grade 1) is Big Trak, a futuristic tank with a small onboard memory and a calculator keypad on the top. Arrowed keys determine the direction the device will move and this command is followed by a number from one to 99. Forward 1 will propel Big Trak forward its own length, and Right 15 will turn it '15 minutes', or 45°, to the right. Up to eight commands can be held at once. It is used as a turtle substitute, encouraging young children to think about programming simple routines. Big Trak demonstrates that commands are carried out in the order in which they were given — a useful initiation into computer programming.

AN OBJECT TO THINK WITH

Although turtles can help to teach mathematics, this is not their primary function; they were designed to teach children how to program a computer. The creator of the turtle, Seymour Papert (see page 506), believes that children should program the computer, rather than allow the computer to program them. This philosophy is clearly evident in Papert's computer language, LOGO, which was designed for learning. He described the turtle as 'an object to think with', whereby a child learns basic programming principles by using the LOGO language to control the turtle's actions.

Paul Cheung, of Edinburgh University, developed the robotic aspect of LOGO, having seen a need for more versatile software at a time when most people were concentrating on hardware development. Cheung wanted children to be able to control a variety of robots. LOGO is ideal for programming turtles, but is limited for programming other devices. Taking LOGO as his starting point, Cheung restructured the language, enabling it to control more functions, and increasing its capacity to receive input from devices such as touch sensors, barcode readers and light detectors. The result was Concurrent—LOGO, or C–LOGO, a language capable of 'parallel processing' (the ability to execute several functions simultaneously).

C–LOGO can control up to eight processes at once. Two particularly interesting functions are FOREVER and WHENEVER. The first, FOREVER, will start a process and allow it to run until C–LOGO terminates it. WHENEVER waits for a condition to arise and then activates a particular function, and is essential for the parallel processing to function. C–LOGO allows switches to be turned on and off and will also detect whether a switch is on or off. This capability is used with the sensor — if a touch sensor presses against an obstruction, it pushes a switch 'on', informing the computer that contact has been made. Children can program a buggy to avoid obstacles, using WHENEVER to tell the robot to turn whenever contact is made.

C–LOGO is more suitable for children in secondary school, where Paul Cheung has used it with a variety of control devices. Children between

the ages of 13 and 15 have written programs to control windmills, lifts, mechanical arms and buggies, as well as a program to interface with a button box that controls the movements of doors and windows in a robotic house. This house contained a working burglar alarm that was activated by the opening of a window, and doors could be opened only after a password was typed into the computer. These two programs were run simultaneously by incorporating parallel processing.

The three floors of the house were accessible by the lift, which was programmed to stop at each. However, the computer must know which floor the lift is at in order for it to function properly. Built from a Meccano kit, the lift was powered by a DC motor with three reed switches fitted to it that detected if the lift was at a particular floor. The children then programmed it to move to the ground floor by pressing one button (on the button box), another for the second floor, and one button for the top floor. They posed themselves the question: 'What will happen if a button is pressed when the lift is moving?" Their immediate answer was to command it to ignore all signals until it had stopped. But after thinking about it, the children decided that a better solution would be to have the computer keep a record of the instructions and implement them in sequence.

SIMPLE SOLUTIONS

It was the features of C–LOGO that enabled them to write what would have otherwise been a complex program, in a simpler fashion. A signal detector, a scheduler and a lift controller were defined as the necessary components, with the signal detector and lift detector acting as parallel processes. Signals were detected using FOREVER, and were then passed to the scheduler. The lift controller then sent a message to the scheduler asking for a destination, which moved the lift to the appropriate floor, and then asked for another destination. After storing the commands in sequence, the scheduler sent them to the controller in order. Inspired by having the lift in front of them, as well as the means to control its programming, the children persevered until the problem was solved.

Many educationalists believe that one of the benefits of children programming robots is that certain 'bugs' can be worked out of their thinking processes. The experience of trial and error, and the consequent development of a more sequential method of planning, shifts the learning condition from a 'right or wrong' mentality, to the more realistic environment requiring constant amendments and changes to suit the given situation. Teachers incorporating the use of robots today see the same potential as those teachers in the early 1980s who first used computers. The appearance of LOGO and C–LOGO, the falling price of hardware and the increasing commercial interest in robotics, indicate a bright future for the use of robots as an educational aid.

Playing Turtle

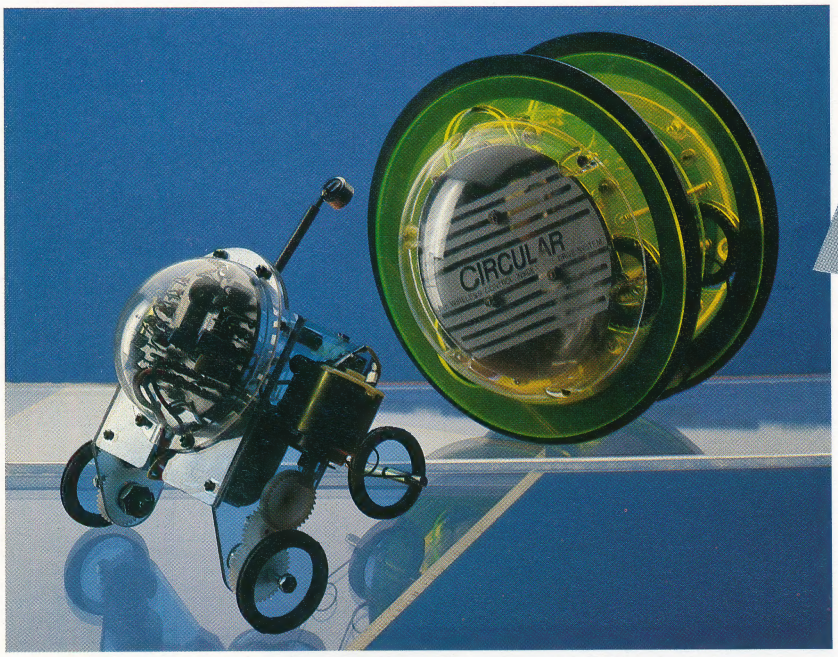
The role of the turtle in the classroom is not limited to geometry, trigonometry and programming. Mazes have been constructed as part of a project on mythology (Theseus and the Minotaur) and on church architecture (the maze set in the floor of Rheims Cathedral). One particularly ambitious project involved the construction of a model town in art and craft lessons, which the turtle had to navigate, visiting a number of different shops in order to 'purchase' goods on its 'shopping list'.

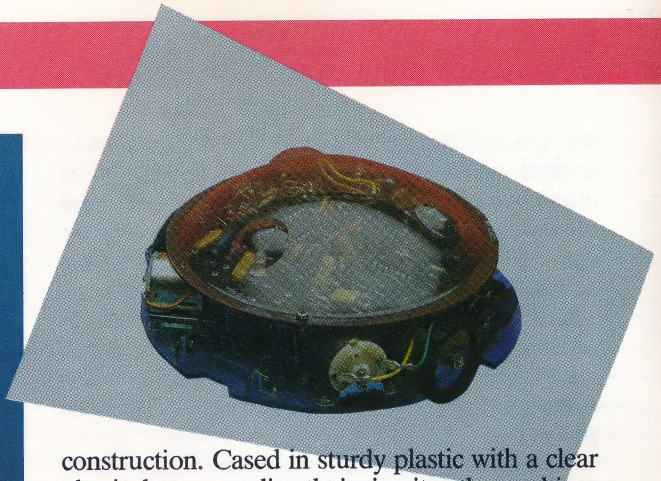
Other turtle games have included Racetrack, where each child must complete with the others in programming the turtle to compete a simple race-course, and Postman, where the children sit in a circle and send written messages to each other, carried by the turtle.

Two games specifically designed to help teach angular displacement and distance are Turnturtle and Shoveturtle. In Turnturtle, the turtle is placed in the centre of a circle divided into sections. Each section is marked with a different number (score) and the child must program the turtle to rotate and come to rest pointing at one of the sections. The child who can get the highest score in the minimum number of programming commands is the winner. Shoveturtle uses a similar method but with a straight track.

Making Tracks
Big Trak is a self-contained unit that can be programmed LOGO-fashion using its numeric keypad. Commands entered consist of a direction (left, right, etc.) followed by the number of steps (up to 99). Big Trak can remember and sequentially execute up to eight different commands

JAN McKINNELL





On The Move
Shown here are the three most expensive 'robots' in the Movits range. From left to right are the Piper Mouse, the Circular and Memocon Crawler. Each machine in the range is cased in attractive transparent plastic, which reveals the internal workings — the motors and electronic components that drive the devices — and gives the machines a suitably 'futuristic' appearance

ROBOTS BY NAME

The increasing technological advances in robotics have generated a wide public interest and a consequent demand for products. The Movits is a series of inexpensive, pre-programmed robot kits that is attempting to cater for this potential market, although whether or not they are actually robots is something we'll discover.

It is widely predicted that robotics will be the next step in the evolution of the computerised home. The stage of development that robotics has now reached is being compared to that of the microcomputer in the early 1970s, with newly emerging technologies combining to form an entirely new product. Whereas the microcomputer was developed in relative obscurity, with only a handful of electronics enthusiasts realising the potential of the new machines, many more people are now aware of the current microelectronics revolution.

Given the wide interest in this technology, it was only a matter of time before an enterprising manufacturer came along with a series of budget-priced 'robots' for the home. The Movits, manufactured by the Kaho Musen Company of Japan and costing between £10 and £35, are among the first attempts to exploit public interest in robotics.

Each Movit comes in kit form with no extra parts needed, apart from 1.5v batteries, and included with some is a screwdriver to help with construction. Cased in sturdy plastic with a clear plastic dome revealing their circuitry, the machines have an exciting, futuristic appearance. There are currently five Movits available, the cheapest of which is the Monkey, which is controlled by sound. On being activated through the sound sensor — 'hearing' a shout or a handclap, for instance — two alternately moving arms pull the device along a string or some other long narrow object. The sensor is a condenser microphone transistor connected via a printed circuit board, which decodes the signal. Appropriate signals will start up the DC electric motor, which turns a crank mechanism and moves the arms. A preset timer on the board ensures the Monkey will stop.

The second Movit in the price range is the Line Tracer II. Retailing at £17.99, the robot is controlled by an infra-red sensor attached to its base, and will follow any high-contrast line drawn on the floor (see page 642). This robot is propelled by two DC motors, generating power for its three wheels. Slightly more expensive at £19.99 is the Piper Mouse, which is also controlled via a sound sensor, this time governed by a 'dog whistle'. Following each burst on the whistle, the Movit will execute one of a sequence of preset commands: turn left, stop, turn right, stop, advance, stop. Following this sequence, the Movit will move in a straight line.

The Circular Movit at £29.99 is the first of the machines with movement governed completely by remote control. The radio-controlled Circular is driven by two DC motors that drive the large wheels on the outside of the bubbles. There are two buttons on the box containing the control; each governs one of the motors. Thus, either the left or right button should be pressed to turn the Movit, and both buttons are pressed to move it forward. For independent control there are two PCB boards, each taking a different signal from the control box.

IS IT A ROBOT?

All the machines so far described are amusing and interesting to build, but the technology they use can hardly be described as advanced. The most 'advanced' of these robots is the Circular, but even this technology has been around for some time, most easily seen in the radio-controlled boats and aeroplanes seen in the parks of cities throughout the world. The crucial point of what we understand to be a robot is that it should have some kind of on-board guidance system which can

be programmed to perform a sequence of actions.

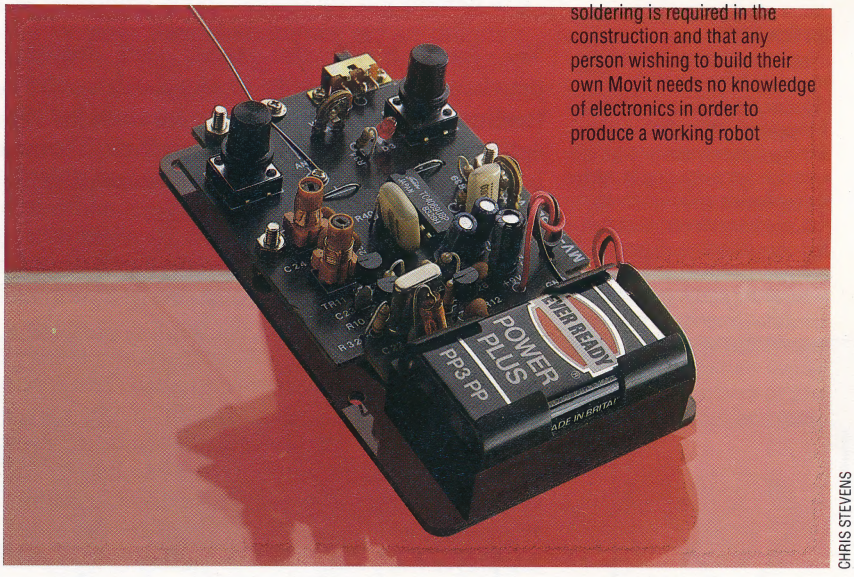
The Memocon Crawler is a machine closely matching our definition of what a robot should be, and like other Movits, is powered by DC electric motors. The guidance system consists of a keyboard connected to the Crawler with a ribbon cable. The keyboard possesses five separate switches for each of the five commands available: forwards, left, right, pause, and sound a buzzer and switch on the machine's LEDs. These commands are stored on a static RAM chip with a memory containing 256 bytes, with each byte consisting of four bits. The bytes holding the instructions are not individually addressable and have to be accessed in sequence. This means that steps within the sequence are not repeatable, although, of course, you can replay the entire sequence again.

Because they come in kit form and need assembling, many people might be disinclined to buy a Movit. However, despite the apparently bewildering number of parts that are provided with each kit, the machines are in fact remarkably easy to construct, and can be built even by children. In fact, a senior executive of a major distribution firm remarked that his ten-year-old son was the main assembler of the company's demonstration models.

The various parts of the robots come in separate bags that are clearly numbered. The instructions consist of an exploded drawing of the Movit. Each part is separately drawn and clearly indicated by its bag number, and arrows point exactly to where each part should go. Most people will find this enough to help them complete the robot, but there are also written instructions to aid you with any part of the construction that may be confusing. Because the printed circuit board is already assembled, the only tools required will be a pair of pliers, a screwdriver and a small hammer.

Movits are undoubtedly fun to build, and owners will derive a great deal of satisfaction from seeing a robot that they themselves have constructed moving about on the floor. But it is doubtful that the Movits are really an entry into the world of robotics. In truth, the majority of these devices are merely sophisticated electronic toys. However, they are inexpensive and although they may not teach you much about robotics, they could easily provide some people, children in particular, with an initiation into this very interesting and increasingly popular subject.

Round In Circles
The Circular, like the rest of the range, is easy to build. The printed circuit boards for the robot itself and the remote control box are already assembled. This means that no soldering is required in the construction and that any person wishing to build their own Movit needs no knowledge of electronics in order to produce a working robot



CHRIS STEVENS

MONKEY
PRICE
£9.99
MOVEMENT
Two alternatively moving gripper arms driven by crank motion
CONTROL
Sound sensor
POWER SOURCE
Two 1.5v batteries

LINE TRACER II
PRICE
£17.99
MOVEMENT
Three wheels driven by two DC motors
CONTROL
Infra-red sensor
POWER SOURCE
Two 1.5v batteries and one 9v battery

PIPER MOUSE
PRICE
£19.99
MOVEMENT
Three wheels driven by two DC motors
CONTROL
Supersonic sound sensor
POWER SOURCE
Two 1.5v batteries and one 9v battery

CIRCULAR
PRICE
£29.99
MOVEMENT
Two cog-geared wheels driven by two DC motors
CONTROL
Radio control box
POWER SOURCE
Three 1.5v batteries, one 9v battery and one other 9v battery for the control box

MEMOCON CRAWLER
PRICE
£34.99
MOVEMENT
Three wheels driven by two DC electric motors
CONTROL
Programmable memory
POWER SOURCE
Two 1.5v batteries and one 9v battery

LISTED COMMODITIES

The New World
The great voyages of discovery took place following the Renaissance at the end of the 15th century. In the 16th century, European explorers were only just beginning to map the continents of the world. At first, these charts were sketchy and often islands — or even entire continents — were placed in the wrong positions. As our simulation progresses, we shall be revealing a different part of the map of world, just as the early explorers pieced the globe's land masses together

We have hired the crew and bought provisions for our ship in our 16th century trading game. All that now remains, before setting sail for the profit-laden New World, is to purchase just a few more things essential to a successful voyage.

The crew's wages will be paid when and if the ship returns to port. The player must decide how much to invest in tradeable goods, hoping to make a profit, and how much gold to keep aside — if any— to help pay the crew's wages on return. The goods available are medicine, guns, salt, cloth, knives and jewels; and the player is informed of their prices and asked the amount to be purchased. The balance of money is printed after each transaction. If there are insufficient funds to pay for the order, the player will be informed and asked to try again. The ship's hold will contain any amount of goods. Once they have been purchased, the voyage can begin. Before the ship sets sail, lists of the crew's composition, the provisions and goods purchased, and the balance of gold are printed.

As with the two previous stages of preparation, the arrays for the third stage, representing the names, prices and amounts of purchased goods are DIMensioned at the beginning of the program. These are added to the previous initialising statements, starting at line 30. The number of each type purchased will be held in array OA(). The descriptions of the other goods are held in the array D$() DIMensioned at line 31. The elements of this array are set in the next two lines. The first, D$(1), is set to BOTTLE OF MEDICINE. The remaining five elements, D$(2) to D$(6), are set to GUN, BAG OF SALT, BALE OF CLOTH, KNIFE and JEWEL. As before, referring to an object by its array address will make the subsequent programming easier.

The cost of these goods is DIMensioned in an array, OC(), containing six elements at lines 34 and 35. The first element, OC(1), represents the price of medicine, which costs one gold piece per bottle. Guns cost five gold pieces (OC(2)=5), and one gold piece will buy five bags of salt (OC(3)=0.2). A bale of cloth costs 2 gold pieces,(OC(4)=2), knives are half a gold piece (OC(5)=0.5) and jewels cost one gold piece each (OC(6)=1).

In the main program, line 500 calls the routine for crew hire, and line 550 calls the provisioning subroutine. Line 600 should now be inserted to call the subroutine that deals with the purchase of other goods. Let's consider this subroutine (lines 3000-3999) in detail. After the screen is cleared there is a delay, and lines 3005 and 3010 print and underline the heading. The player is then told that other commodities may be needed for the voyage.

The programming for the purchasing of the six types of goods is organised as a loop. The loop is used once for each type using T as the loop counter from 1 to 6. The player is told the cost of each item, and its name is represented by D$(T) in line 3070.

The first time round the loop, T will be set to 1, for the first item — medicine. The unit cost of the current trading item is set by OC(T) in line 3080. When T is set to 1, OC(T) will represent one gold piece, which is the cost of a bottle of medicine. The second time around the loop, T will be incremented to 2, and D$(T) will be set to GUN and the cost, OC(T), will be five gold pieces, and so on, until (T) reaches 6.

TEXT FORMAT

If the item costs more than one gold piece, an S must be added to GOLD PIECE, which is printed by line 3081. If the cost is one gold piece, then a space is printed after PIECE; if it is more, an S is printed. There follows a delay, and the player is asked, 'WOULD YOU LIKE TO BUY? (Y/N)'. Line 3110 waits for a reply, P$. The left-most character of the reply is checked by line 3120. It is sent to line 3175 if it is 'N', sending the program back to the start of the loop where the next item is presented. If 'Y' or 'YES' is input, there is a delay, and line 3130 asks 'HOW MANY DO YOU WANT?'. Line 3135 waits for a response, and you could answer by typing 10, or 10 SACKS OF SALT. This choice is available because line 3140 looks at the reply and uses VAL to acknowledge the numeric characters at the beginning of the phrase and ignore the text.

If you try to make an illegal purchase, not having enough gold to pay the bill, line 45 checks to determine if the cost of the item, OC(T), multiplied by the number input, TT, is greater than the remaining money, MO. If this is the case, the program is sent to 3150, which informs the player of the lack of funds. After a delay, the player is told 'PLEASE ENTER AGAIN' in line 3154. The program returns to line 3130, which asks 'HOW MANY DO YOU WANT?' if the new request can be paid for. If there are enough funds, the program goes to line 3160, where the amount spent is subtracted from the kitty, using the formula: MO = MO — (OC(T)*TT). MO represents the money remaining. The cost of the item, OC(T), is multiplied by the amount purchased, TT, and subtracted from the balance. TT is stored in array OA() at line 3165, and after a delay, a blank line and the balance of gold is printed by line 3176. After another delay, the value of T is increased by one, and the program is returned to the start of the loop by line 3200, where it presents the next item for purchase.

When all the items have been selected — and after a pause — line 3210 tells the player that the purchasing stage is over. Line 3230 prints 'PRESS ANY KEY TO CONTINUE' using the variable set up at the start of the program, K$, and waits for a reply before returning to the main program. The final task is to summarise the goods ordered and crew hired before we move on to start the voyage. The Opening Summary Routine does this.

The program prints, 'YOU HAVE THE FOLLOWING CREW', and the crew types are printed in a loop, at line 645 (FOR T=1 TO 5) for each of the five types of crew member. Line 650 checks if any of that type have been hired, by checking CC(T), the crew count for that type. If the count is zero, line 670 sends the program back to the start of the loop, increasing the value of T by one. If any of that crew type have been hired the number and category are printed. An S or a space is printed after the description, the program looping five times for each crew type. A list of provisions is printed in a similar way, from lines 675 to 710, using a loop of 4 (line 685) for the four provision types.

SPECIAL CONSIDERATIONS

The remaining goods bought are not printed in a loop because their descriptions are so different. They are not described in units, so simply adding an 'S' will not necessarily create a pluralised word; knife becomes kni*ves* and a bale of cloth becomes bale*s* of cloth, for instance. Each of the extra goods is dealt with separately in a linear fashion. If a type of merchandise hasn't been bought, the program goes straight to the next type. Line 730, for example, checks if medicine has been bought, and if not the program jumps to the next section at line

Tools Of The Trade
Here are the goods that our explorers might have been expected to take with them to trade with the inhabitants of the new lands. Of course, they will have been expected to pay for the articles before they set sail with them. In our simulation, the names of each of the possible articles carried are held in the string D$. The price per item is contained in the array OC, while the number purchased is held in the array OA

IAN McKINNELL

IAN McKINNELL

740. If medicine has been bought, the program determines if it was one or more bottles and prints the appropriate ending — an S or a space. It then prints the amount and goes onto the next type.

When the list is complete, line 792 informs the player of the amount of money remaining. The player is told 'PRESS ANY KEY TO START VOYAGE' at line 797, and the program then waits for an input.

Module Three: Trade Goods

Array Dimensioning

```
30 DIMOA(6)
31 DIMD$(6)
32 D$(1)="BOTTLE OF MEDICINE":D$(2)="GUN":D$(3)="B
AG OF SALT"
33 D$(4)="BALE OF CLOTH":D$(5)="KNIFE":D$(6)="JEWE
L."
34 DIMOC(6)
35 OC(1)=1:OC(2)=5:OC(3)=.2
36 OC(4)=2:OC(5)=.5:OC(6)=1
```

Call To Trading Goods Subroutine

```
600 GOSUB3000
```

Opening Summary Routine

```
605 REM **** READY TO START ****
610 PRINTCHR$(147)
615 S$="YOU ARE NOW READY TO START*":GOSUB9100
625 S$="THE VOYAGE.*":GOSUB9100
630 GOSUB9200
635 PRINT:S$="YOU HAVE THE FOLLOWING CREW:*":GOSUB
9100
640 GOSUB9200
645 FORT=1TO5
650 IFCC(T)=0THEN670
655 PRINTCC(T);
660 PRINTC$(T);
662 IF CC(T)=1THENPRINT" ":GOTO668
664 PRINT"S"
668 GOSUB9200
670 NEXT
674 GOSUB9200
675 PRINT:S$="AND THE FOLLOWING PROVISIONS:*":GOSU
B9100
680 GOSUB9200
685 FORT=1TO4
690 IFPA(T)=0THEN710
695 PRINTPA(T);U$(T);"S OF ";
700 PRINTP$(T)
708 GOSUB9200
710 NEXT
715 GOSUB9200
720 PRINT:S$="YOU HAVE ALSO GOT:*":GOSUB9100
725 GOSUB9200
730 IFOA(1)=0THEN740
733 IFOA(1)=1THENS$="BOTTLE OF MEDICINE *":GOTO
735
734 S$="BOTTLES OF MEDICINE*"
735 PRINTOA(1);:GOSUB9100
736 GOSUB9200
740 IFOA(2)=0THEN750
743 IFOA(2)=1THENS$="GUN*":GOTO745
744 S$="GUNS*"
745 PRINTOA(2);:GOSUB9100
746 GOSUB9200
750 IFOA(3)=0THEN760
753 IFOA(3)=1THENS$="BAG OF SALT*":GOTO755
754 S$="BAGS OF SALT*"
755 PRINTOA(3);:GOSUB9100
756 GOSUB9200
760 IFOA(4)=0THEN770
763 IFOA(4)=1THENS$="BALE OF CLOTH*":GOTO765
764 S$="BALES OF CLOTH*"
765 PRINTOA(4);:GOSUB9100
766 GOSUB9200
770 IFOA(5)=0THEN780
773 IFOA(5)=1THENS$="KNIFE*":GOTO775
774 S$="KNIVES*"
775 PRINTOA(5);:GOSUB9100
776 GOSUB9200
780 IFOA(6)=0THEN790
783 IFOA(6)=1THENS$="JEWEL*":GOTO785
784 S$="JEWELS*"
785 PPRINTOA(6);:GOSUB9100
786 GOSUB9200
790 GOSUB9200
792 PPINT:PRINT"YOU HAVE ";MO;" GOLD PIECES
LEFT"
796 GOSUB9200
797 S$="PRESS ANY KEY TO START VOYAGE*"
798 GOSUB9100
799 GETP$:IF P$=""THEN799
999 END
```

Trading Goods Subroutine

```
3000 REM **** STAGE 3 OTHER GOODS ****
3001 PRINTCHR$(147): REM STAGE 3
3002 GOSUB9200
3005 PRINT"     STAGE 3 - OTHER GOODS"
3010 PRINT"     -------   -----------"
3020 GOSUB9200
3025 PRINT
3030 S$="THERE ARE OTHER THINGS THAT MAY*":GOSUB91
00
3035 S$="BE USEFUL ON THE VOYAGE, FOR  *":GOSUB910
0
3040 S$="EXAMPLE MEDICINE AND TRADING  *":GOSUB910
0
3045 S$="GOODS.*":GOSUB9100
3046 GOSUB9200
3050 S$="YOU MAY ALSO NEED WEAPONS.*":GOSUB9100
3055 GOSUB9200:GOSUB9200
3060 FORT=1TO6
3065 PRINT
3070 PRINT"A ";D$(T);
3075 S$=" COSTS*":GOSUB9100
3080 PRINTOC(T);
3081 PRINT" GOLD PIECE";
3085 IFOC(T)=1THENPRINT" ":GOTO3090
3086 PRINT"S"
3090 GOSUB9200
3095 S$="WOULD YOU LIKE TO BUY (Y/N)*":GOSUB9
100
3110 INPUTP$:P$=LEFT$(P$,1)
3115 IF P$<>"Y"AND P$<>"N" THEN 3095
3120 IF P$="N"THEN3175
3125 GOSUB9200
3130 S$="HOW MANY DO YOU WANT*":GOSUB9100
3135 INPUTP$
3140 TT=VAL(P$)
3145 IFOC(T)*TT>MOTHEN3150
3147 GOTO3160
3150 S$="YOU DON'T HAVE ENOUGH MONEY*":GOSUB9
100
3152 GOSUB9200
3154 S$="PLEASE ENTER AGAIN  *":GOSUB9100
3155 GOSUB9200:GOTO3130
3160 MO=MO-(OC(T)*TT)
3165 OA(T)=TT
3170 GOSUB9200
3175 PRINT
3176 PRINT"MONEY LEFT = ";MO
3200 GOSUB9200:NEXT T
3205 GOSUB9200:PRINT:PRINT
3210 S$="END OF STAGE 3*":GOSUB9100
3220 GOSUB9200:PRINT
3230 S$=K$:GOSUB9100
3240 GETP$:IFP$=""THEN3240
3999 RETURN
```

Basic Flavours

Spectrum:
Make the following changes:

```
 32 DIM D$ (6,20)
610 CLS
799 LET P$=INKEY$: IF P$ = "" THEN GO TO 799
3001 CLS
3110 INPUT P$ : LET P$ = P$ (1 TO 1)
3240 LET P$=INKEY$: IF P$ = "" THEN GO TO 3240
```

BBC Micro:
Make the following changes:

```
610 CLS
799 A$=GET$
3001 CLS
3240 A$=GET$
```

M

MINICOMPUTER

Like 'mainframe' (see page 1049), the term *minicomputer* has become increasingly blurred in the past few years as the advances in computer technology have erased previous delineations. The term minicomputer originally referred to a computer that could be held in a single cabinet about the size of an average wardrobe, whereas mainframes would occupy several times that space. Although slower and less powerful than their larger counterparts, they are considerably less expensive, and can usually be found processing data for medium-sized businesses, where the enormous power and price of a mainframe would be inappropriate.

Some experts have predicted the extinction of the minicomputer as micros become increasingly sophisticated. However, unless there is a dramatic and sudden increase in the power of microprocessors, minicomputers look to be with us for a few years yet.

MODEM

A *modem* (an acronym for *mo*dulator/*dem*odulator) is a device enabling computers to communicate with each other via the telephone network. The machine operates by converting the computer's digital bit patterns into the analogue signal that is employed in the telephone system. This process is known as 'modulation'.

'Demodulation' occurs when the modem receives an analogue signal from the telephone and converts it back into a digital signal that can be understood by the host computer. Although this is the modem's principal function most recent modems also have some capacity for varying the rate at which the bit patterns are transmitted (the baud rate). Because there is not yet a standard transmission speed, many computers transmit data at different rates. Thus, there must be some method of standardising the baud rate of interconnected computers.

IAN McKINNELL

As microcomputer communications develop, and a growing number of databases and similar applications become available to the modem-equipped micro, it seems certain that these devices will come to be as important a peripheral to the serious user as mass storage devices, such as tape decks and disk drives, are now.

MODULAR PROGRAMMING

The term *modular programming* is interchangeable with 'structured programming' (see page 454) and refers to a program that is broken down into various modules, each performing a single task. This is in contrast to the 'monolithic' style of programming. An advantage of this system is that the modules making up the program can be tested independently of each other. When the individual modules are working satisfactorily, they can be connected together in a single, larger program by passing data such as variables between the modules. Another advantage to this approach is that the programmer can build up a library of commonly used modules that can be dropped into a program without alteration, which saves the tasks from having to be rewritten. Although he did not originate the idea, the accepted criteria of modular programming were put forward by the computer theorist David Parnas who said that a modular program should be based on 'information hiding' (see page 816).

MODULATION

This is a method whereby the shape of a particular set of waves within a signal is altered. In computing, the most common form of *modulation* is the changing of a digital signal to an analogue signal, and reversing the process (known as 'demodulation'). Devices that carry out these operations are known as modems. There are several different methods of modulation. 'Amplitude modulation' (AM) refers to the varying of the amplitude (the distance from which the wave moves from state zero) of a wave. 'Frequency modulation' (FM) denotes the altering of a number of waves passing a point within a given time. This is the technique most often used in modems. 'Phase modulation' (PM) occurs when the phase, or the completed cycle, of the wave is varied. This method is used in conjunction with amplitude modulation in certain high speed modems. 'Pulse code modulation' (PCM) is a system with which to read analogue signals as pulses within a digital system.

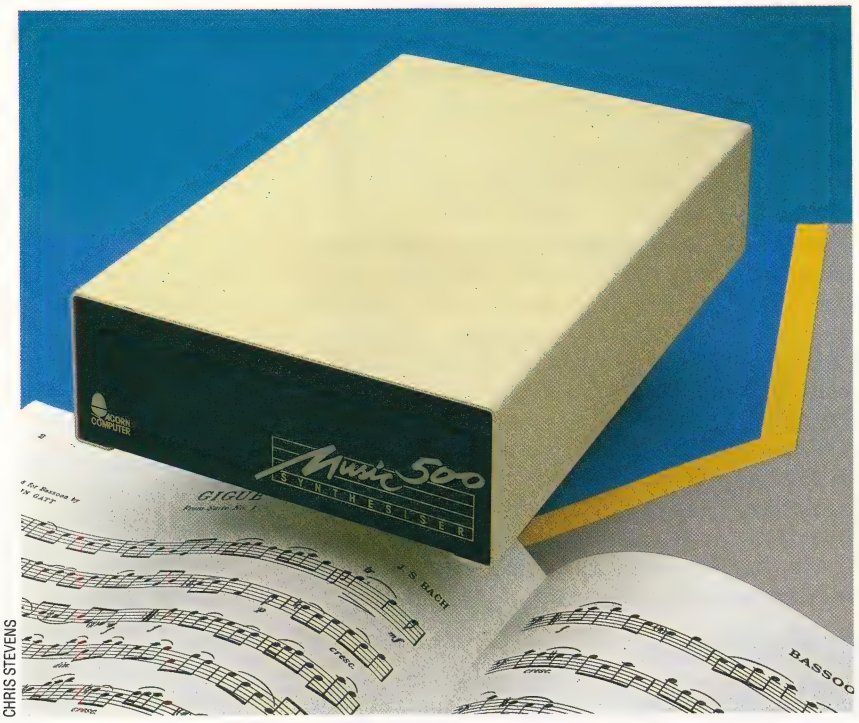
MODULUS

Generally known by its abbreviation 'mod', this is a commonly used arithmetic operation that has been adopted by the PASCAL programming language, in particular. The effect of A MOD B is that the result will be the remainder of the division of A by B. MOD is generally used in conjunction with the DIVide operator, as the effect of DIV produces only an integer result. For example: 17 DIV 5 gives the result 3; 17 MOD 5 produces the result 2 (17 divided by 5 is 3, with a remainder of 2).

On The Line

There are a number of different types of modem currently available. But, broadly speaking, these can be divided into two groups: acoustic modems, where the telephone handset is placed into the modem; and hard-wired modems, in which a lead from the computer is plugged directly into the telephone junction box

AMPLE ETUDES



CHRIS STEVENS

A Little Byte Music
Hybrid's Music 500 package is certainly one of the most advanced computer music devices available for home micros. Equipped with 16 oscillators and digital wave modulation, it is certainly a match for much more expensive synthesisers

Produced by Hybrid Technology, marketed by Acorn and intended for the BBC Micro, the Music 500 incorporates music making technology hitherto unavailable for home micros. With a music programming language, AMPLE, and the imminent release of a 49-note piano keyboard, the Music 500 offers extraordinary possibilities.

The Music 500 Package, marketed by Acorn for the BBC Micro, was designed by another Cambridge company, Hybrid Technology. Priced at £200, the Music 500 is a complex piece of electronics housed in a cream coloured disk drive box. It connects to the BBC Micro's 1 MHz bus connector and to a stereo amplifier via a standard 5-pin DIN socket.

Inside the box are 16 oscillators, each programmable by the host computer for frequency (pitch), volume, balance and waveform. With its accompanying software the Music 500 can out-perform a number of currently available synthesisers many times its price.

Most music synthesisers produce their range of sounds by one of two methods. The cheaper ones take a simple tone — a sine wave — and then distort and filter it to produce the required effect. Although this method can produce a reasonable range of sounds, it is still very limited, even in the hands of an expert. Some of the more expensive synthesisers use several sine waves to modulate one another. The amplitude of one wave is used to control the frequency of another. This method of control is vastly superior to the simple distortion method but is still not able to imitate most naturally produced sounds with accuracy (see pages 581 — 584 for a more detailed description).

The Music 500, however, produces its sounds by an entirely different method. The waveforms created by this device are digitally built up to a programmed pattern and are sampled at a rate of 47,000 times each second. In addition to this ability to program any wave shape, the Music 500 also offers modulation techniques — it is not restricted to frequency modulation, however, and the Music 500 can combine channels with ring modulation and phase modulation. It can also use wave shapes other than the simple sine waves.

Simply creating a tone on the device would be a tricky job if it weren't for the software supplied with the Music 500. This is supplied on cassette though it can be easily transferred to disk with the provided program. Hybrid Technology has plans to release an extended version of the software on ROM in the near future. The package is supplied with AMPLE — a music programming language that provides everything you need to make the most of your Music 500. The language is in many ways like FORTH or LOGO to use, being a word-based language with which programmers can define passwords and procedures. A musical composition might look something like this.

```
10 .%FLIGHT OF THE BUMBLE BEEB
20 .setup
30 .play
```

The word setup is a user-defined word that has already been programmed as follows:

```
10 .'setup'
20 .[
30 .setsynth
40 .setbass
50 .setdrums
60 .setcymbal
70 .]
```

The terms with which setup is defined have in turn been pre-programmed to define different sounds — e.g. *setdrums* or *setcymbal*. Preset waveforms are provided to get you started with the Music 500, however, new waveforms are easily created. Waveforms can be defined either in terms of the relative proportions of the first 16 harmonics of a tone, or 'drawn out' in mathematical form. This latter method gives the Music 500 the ability to produce 'noise' by specifying a random waveform.

The definition for the cymbal sound, for example, might look like this.

```
10 .'cymbal'
20 .[
30 .3 WMOD 0 rand! 128 FOR (RAND? INDEX WG!)
   FOR WGC
40 .1 CHAN
50 .3 WAVE ON RM —1 POS
60 .2 CHAN
70 .3 WAVE 2000 OFFSET 1 POS
80 .]
```

The AMPLE equivalent of a BASIC FOR. . .NEXT loop sets each point in the waveform to a random value. This waveform is used for two sound producing channels (two oscillators) and the pitch of the second is offset slightly from the first and is in a different stereo position. The result is a noisy, crashing sound — like a cymbal.

Alternatively a more tuneful waveform could be defined:

```
10 .'synth'
20 .[
30 .WZERO 181 1 WH! 135 2 WH! 181 3 WH!
          62 4 WH!  76 5 WH!  37 6WH!
          14 7 WH!  3 8 WH!
40 .WHG WGC 3
50 .]
```

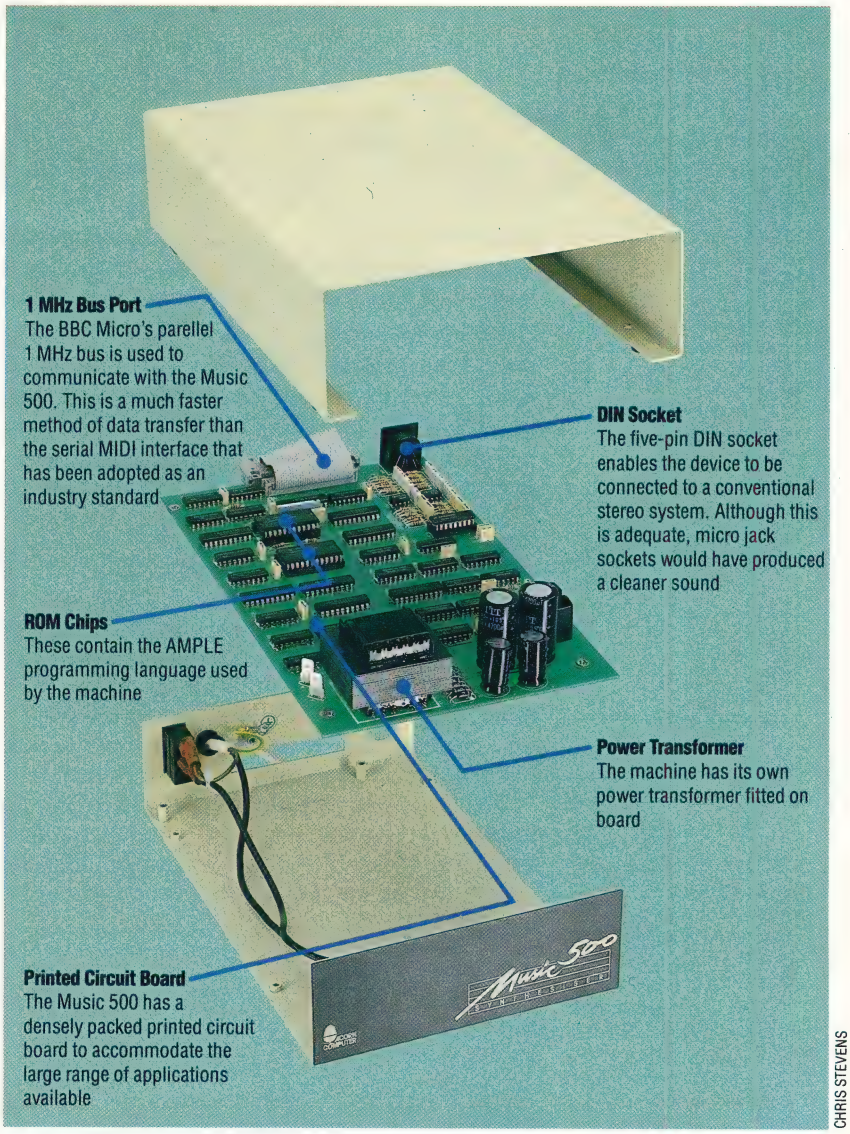
This waveform is made up of varying proportions of the first eight harmonics. The pitch and amplitude envelopes are similarly created, using either the simple default forms or what has been specified by the user.

The word play might call up each of the defined sounds and more words that contain the actual score.

```
10 ."play"
20 .[
30 .4 PLAYERS
40 .1 PLAY (piano melody)PLAY
50 .2 PLAY (bass bassline)PLAY
60 .3 PLAY (drums rhythm)PLAY
70 .4 PLAY (cymbal crash)PLAY
80 .GO
90 .]
```

Simply put, the system is given the number of players needed, each player is given a sound and a score — more user-defined words — and finally they are all told to get on with the music, in time with each other. This multi-tasking ability of AMPLE allows the Music 500 to be used as if writing a score for an orchestra. You don't have to score all the parts at once, though you can if you wish, and each is treated separately. The entire language is modular and hierarchical. Even the built-in words are redefinable, although using this feature can get quite complicated. However, it is possible to, say, redefine the bar marker word so that the second note in each bar is played a little louder for stress.

If all this sounds a bit too technical, then AMPLE provides some encouraging words. Sensible defaults are provided throughout, so if you only want to play a tune then you can just enter the notes at the keyboard rather than inside a word definition.


1 MHz Bus Port
The BBC Micro's parellel 1 MHz bus is used to communicate with the Music 500. This is a much faster method of data transfer than the serial MIDI interface that has been adopted as an industry standard
DIN Socket
The five-pin DIN socket enables the device to be connected to a conventional stereo system. Although this is adequate, micro jack sockets would have produced a cleaner sound
ROM Chips
These contain the AMPLE programming language used by the machine
Power Transformer
The machine has its own power transformer fitted on board
Printed Circuit Board
The Music 500 has a densely packed printed circuit board to accommodate the large range of applications available
Music 500

CHRIS STEVENS

Acorn is soon to release a full 49-note musical keyboard that connects to the BBC Micro's user port, for use with the Music 500. This will include AMPLE software such as a 'note processor' that will record your keyboard playing and then allow you to replay and edit it.

MUSIC 500	
Price	£199 inc VAT
Oscillators	16
Frequency range	0-20 Khz
Pitch resolution	16th semitone
Waveforms	
Harmonic	16 harmonics
Geometric	128 points
Envelopes (pitch/amplitude)	17 segments
Step length	0-320 seconds
Time resolution	10ms
Stereo positions	7
Waveform precision	8-bit logarithmic
Sample rate	46875/sec
Frequency resolution	0.0056 Hz
Time base	255ms-655ms/cycle
Concurrency	1-9 processes

PLAY IT AGAIN, RAM

Thus far in our PASCAL series, we have looked at most of the language's rich set of structured statements, enabling us to write programs to solve most types of problems. Here we examine the three different 'iteration' — or repetition — constructs, which allow us to control program loops.

We have already looked at two of PASCAL's three statement structuring methods: sequential structures using the 'word brackets' BEGIN and END (see page 1024), and choice, or selection, using IF and CASE constructs (see page 1046). The third type of statement structure is 'iteration', or repetition, for which PASCAL provides three different constructs. The FOR statement gives a 'counter-driven' loop, whereas both WHILE and REPEAT statements are 'condition-driven'.

The Boolean expression delimited by the reserved words WHILE and DO is evaluated first, and if the result is true, the following statement (which can be any PASCAL statement including a structured statement of any degree of complexity) will be repeatedly executed as long as the condition remains true. This Boolean condition is re-evaluated after every execution of the statement in the 'body' of the WHILE construct. Obviously, this implies that at least one of the items evaluated in the expression must be changed in some way by the actions within the loop. As an example:

```
WHILE MaxInt > 1 DO
   WriteLn ( 'Looping!' )
```

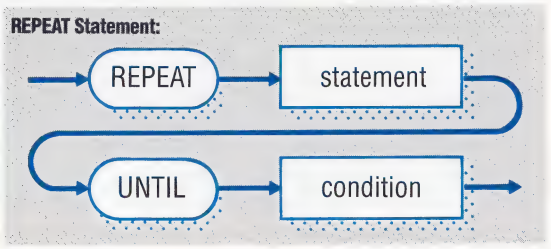
would have great difficulty in terminating. Here's a more realistic (and safer) example:

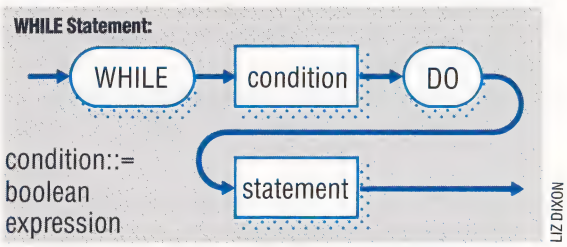
```
read ( withdrawal );
WHILE withdrawal > BankBalance DO
   BEGIN
      WriteLn ('Insufficient funds — try again:');
      write ('Amount ?');
      read (withdrawal)
   END
```

The most valuable asset of the WHILE construct is well illustrated by the previous program segment. If, when the withdrawal amount is first read, it is within the current balance of the account, the WHILE loop will not be performed at all. If the body is executed, looping will stop when the expression becomes false. The flow of control reverts to sequential operation, and the first statement after the WHILE construct will then be executed. The appearance of the Boolean expression at the head of this construct indicates syntactically the security of the PASCAL 'semantics' (or the precise meaning and behaviour of the statements). If you want to iterate only if a condition is true, the WHILE loop should be used.

There are often situations when we need to perform the body of a loop at least once in order to set up the condition that stops it, as with the Golden program example (see page 1076). Under these circumstances the REPEAT . . UNTIL construct more accurately expresses the algorithm, and provides a natural high-level syntax.

Notice that the terminating condition now appears at the end of the structure, and that it is logically opposite to the condition used to code the WHILE construct. That is to say, the REPEAT loop terminates when the Boolean expression evaluates to true, whereas a WHILE construct stops (if indeed it ever started) on the condition being false. If the body of the REPEAT statement contains several statements, they should really be enclosed within BEGIN . . END word brackets as a compound statement, but PASCAL relaxes this rule because the reserved words REPEAT and UNTIL will sufficiently delimit the loop's body. It's perfectly legal to put the superfluous BEGIN and END words in, however, rather like using an extra semi-colon before a reserved word except that no null statement is created.





LIZ DIXON

The following segment will count the number of occurrences of the letter 'e' in a sentence entered from the keyboard. You must complete the sentence with a full stop when the REPEAT loop terminates and the result is printed.

```
count := 0;
REPEAT
   read (symbol);
   IF symbol = 'e' THEN
      count := count + 1
UNTIL symbol ='.';
WriteLn ('There were', count : 1,
   '''e''s in the sentence.')
```

In this case, there is no obvious preference for either construct, and we could just as well have used a WHILE loop:

```
count := 0;
read (symbol);
```

Golden Exercise Answer

On page 1076, we set an exercise with our Golden program, which generated Fibonacci terms and the ratios of successive pairs. We asked you to try to alter the loop so that the terminating condition stops when the next Fibonacci term to be calculated exceeds MaxInt.

With two's complement representation, if an integer result exceeds MaxInt, the sign bit gets 'clobbered'. The number then looks negative, so our Golden program would crash, as the type Fibonacci excludes negative values. Even with integers, the loop would never stop. We can't say:

UNTIL first + second > MaxInt

because there is no number greater than MaxInt, by definition. However, a little cunning rearrangement to do subtraction rather than addition provides the solution:

UNTIL second > MaxInt—first

Now, of course, the real arithmetic and the constant Epsilon are no longer required

```
WHILE symbol <>'.' DO
  BEGIN
    IF symbol = 'e' THEN
      count := succ (count);
    read (symbol)
  END;
write ('There were', count : 1, );
WriteLn ('''e''s in the sentence.')
```

The difference between the two structures can clearly be seen, however. The action required to set up the terminating condition (the read statement in this example) will often occur twice when a WHILE construct is used, once immediately before entry to the construct and again as the last statement in the body of the WHILE loop.

Although the WHILE loop is the only essential construct in any language, it is sometimes convenient to have a means of repeating something a specified number of times, perhaps over a certain range of values within a scale. PASCAL provides a third construct for creating such 'counter-driven' loops. The BASIC programmer may feel on familiar territory here, but there are a number of important differences between the PASCAL construct and BASIC's FOR loop that should be borne in mind. First, any scalar variable may be used as the loop controller, not just integers. Secondly, and more importantly, the PASCAL FOR . . DO statement is completely secure; like the WHILE loop, it may not be executed at all (if, for instance, the starting value is greater then the final one) and there are severe restrictions placed upon the use of the controlling variable in PASCAL. In particular, it is illegal to change this value anywhere in the body of the loop. Moreover it is an error in PASCAL merely to threaten to change it! We'll see the real benefit of these safety precautions when we deal with procedures and functions, but meanwhile, here is an illegal example:

```
FOR N := 1 TO 10 DO
  IF N = 10 THEN
    N := 1
```

Sheer silliness, of course. Having requested the loop to be performed ten times, the IF statement would reset the value of N back to 1 and create an infinite loop.

Notice that the syntax of the FOR . . DO construct has an assignment statement between the first two delimiters (FOR and TO) which assigns the initial value to the loop control variable, and the final value of the scale is given by the expression enclosed by the reserved words TO and DO. Both these values must, of course, be the same simple scalar type as the controller. Some further examples:

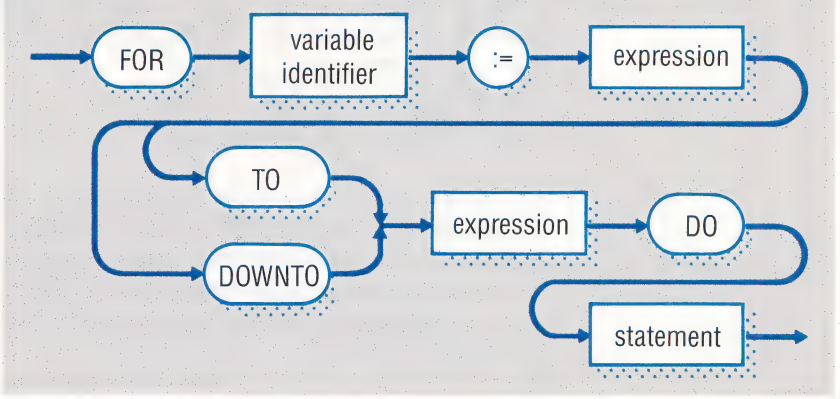
- FOR letter := 'A' TO 'Z' DO { etc }
- FOR month := Jan TO Dec DO { etc }
- FOR N := N TO succ (MaxInt DIV 1000) DO { etc }

The last example assumes that N has been given a suitable value some time previously, and remember that if this were to exceed the specified final value, the loop would not be performed at all. For the occasions when we wish to descend a scale of values, the reserved word DOWNTO is used in place of TO. So, for example, NASA might be able to use:

```
FOR CountDown := 10 DOWNTO 0 DO
  WriteLn (Countdown : 32 — 3 * CountDown );
WriteLn ('We have LIFT OFF !')
```

PASCAL maintains rigorous control over the FOR loop, and does not allow any equivalent of BASIC's optional STEP increment or decrement.

FOR Statement:



Real Ale Program

```
PROGRAM       RealAle   ( output );

TYPE
  beer = ( Mild, Bitter, Special, Warmer );
VAR
    price,
    pints  : real;
    ale    : beer;
BEGIN
  WriteLn ( 'Pints' : 6,
      'Mild' :9, 'Bitter' :8,
        'Special' :8, 'Warmer' :8 );
  WriteLn;
  pints := 0.5; { start at half a pint }

  REPEAT
    IF pints - trunc ( pints ) > 0.4
      THEN { write as ##.# }
        write ( pints : 6 : 1, ' ' )
      ELSE { write as an integer }
        write ( round ( pints ) :4, ' ':3 );

    FOR ale := Mild TO Warmer DO
      BEGIN
        CASE ale OF
          Mild    : price := 65;
          Bitter  : price := 69;
          Special : price := 74;
          Warmer  : price := 79
        END;    { CASE }
          { round and convert to pounds : }
        write ( round ( price * pints )
                          / 100 : 8 : 2 )
      END;
        { now start a new line }
    WriteLn;
    pints := pints + 0.5

  UNTIL pints > 10
END .
```

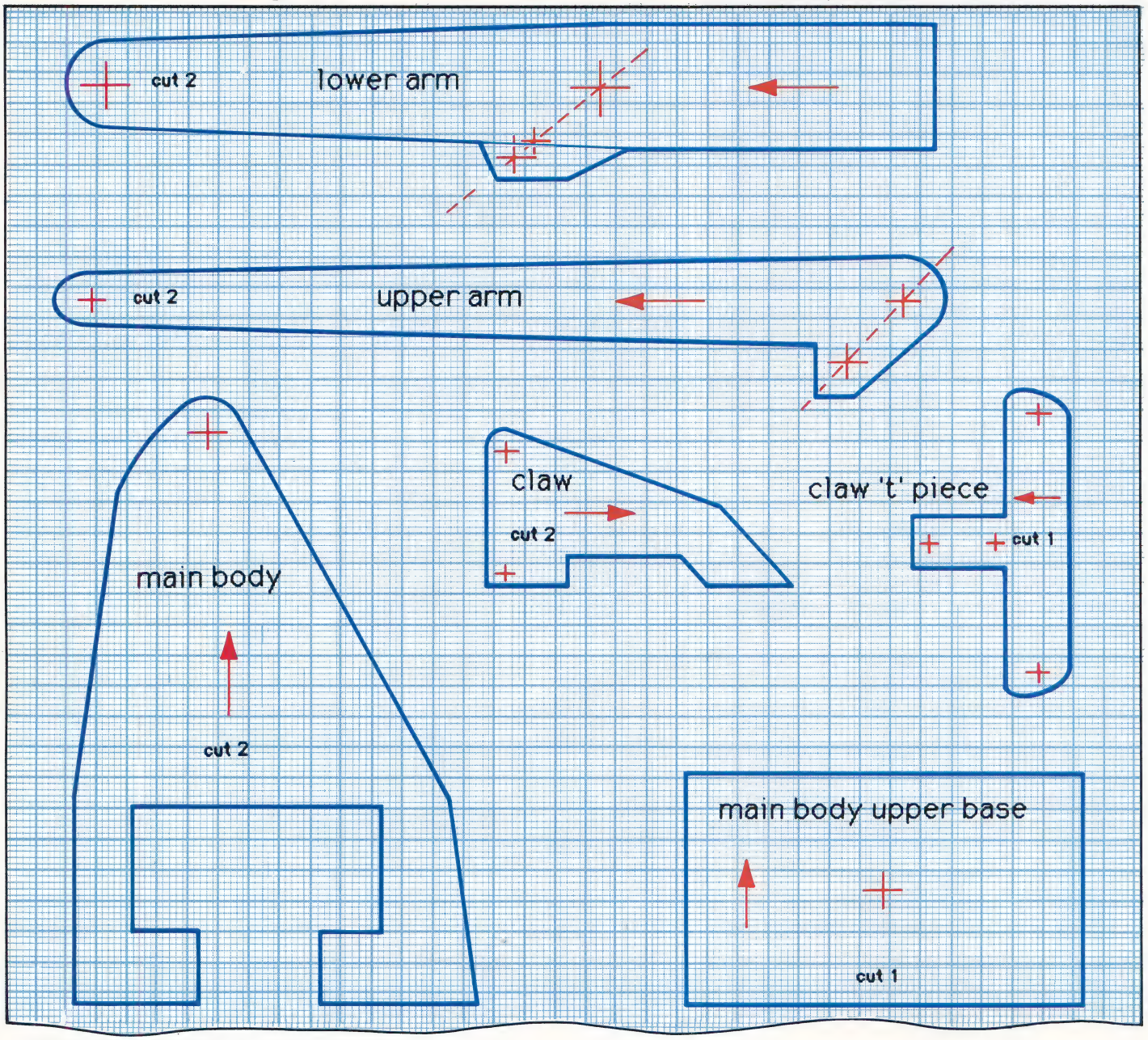
What's Yours?
The program Real Ale prints a price list of four different types of beer for a range of quantities in half pint increments. Each class of ale is represented by a conceptual constant identifier enumerated in the TYPE definition of beer. The selection of the associated price is naturally made with a CASE statement. Whole pints are printed as integers, padded out with two extra spaces, while the formatting of real amounts suppresses unwanted decimal places

STRONG-ARM TACTICS

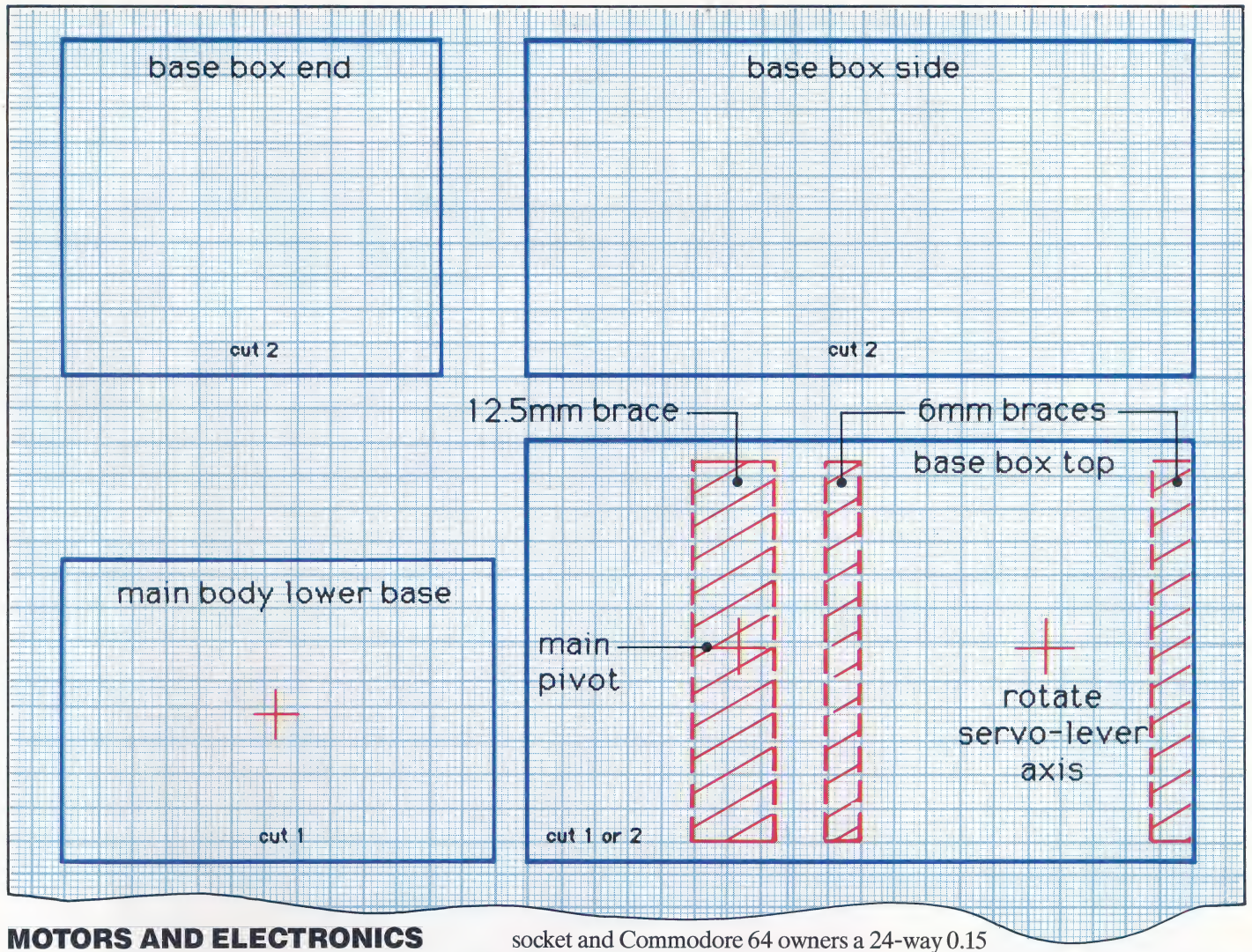
We begin the construction of the Workshop robot arm by providing the cutout patterns for the arm members, as well as detailing the rest of the parts required to complete the project.

The parts needed for this project will probably have to be assembled from several sources. Electronics and modellers' shops should be able to provide the electronic and moving parts, other parts being available from hardware and DIY shops. The total cost of building the project will vary depending on the source of the parts but should be between £70 and £80.

The base and arm members must be constructed from a light, rigid material that is easily cut and shaped. The most accessible material is probably plywood, although a thin sheet of perspex, or rigid plastic material, could also be used. Assuming that plywood will be used to construct the frame, we require a 50 by 50cm square, 4mm thick. We will use this in conjunction with patterns to cut out the frame parts. Three 55mm lengths of wood (two 6mm thick and one 12.5mm thick) are required to act as braces.



KEVIN JONES



MOTORS AND ELECTRONICS

We need four 5v digital servos for the arm — Futaba S128 motors are ideal for the purpose. 30mm plastic discs should be obtained to fit in the end of the motor spindles, which should be obtainable from the same source as the motors. Modellers' shops are probably the best place to try. If you want a more powerful arm then you could use more powerful motors than the Futaba S128.

To power the motor we need a 5v DC supply. Although all three micros that we'll be using to control the arm have 5v supplies at the user or expansion port, these voltages are designed to drive external electronic circuits at TTL levels. The maximum current that should be taken from these 5v sources is 100mA. Because a servo motor under full load can draw 200mA it is inadvisable to power the arm servos from the user port. We therefore need an external power source. Ideally this would be a 5v DC transformer, but a less expensive alternative is to use a battery clip with three 1.5v batteries. The total output of 4.5v will be sufficient to drive the system.

A 9-hole by 20-strip piece of veroboard is required to mount sockets for the motors to plug into, and a 2m length of 20-way ribbon cable will be needed to connect the arm to the computer. BBC Micro owners should buy a 20-way IDC socket and Commodore 64 owners a 24-way 0.15 inch edge connector, to connect the arm to the user port. The arm will be linked to the Spectrum via the interface built earlier in Workshop to control the robot (see page 992). We will detail the necessary modifications to this interface in a future instalment.

Three 75mm lengths of brass tubing with an outside diameter of 4mm are required together with three 75mm lengths of 5mm diameter brass tubing to make the arm pivots. Each pivot will be made by placing the 4mm tube inside the 5mm tube and trimming the ends to size. It is therefore important that the tubes you buy fit snuggly together, but with sufficient clearance so the inner tube can rotate freely.

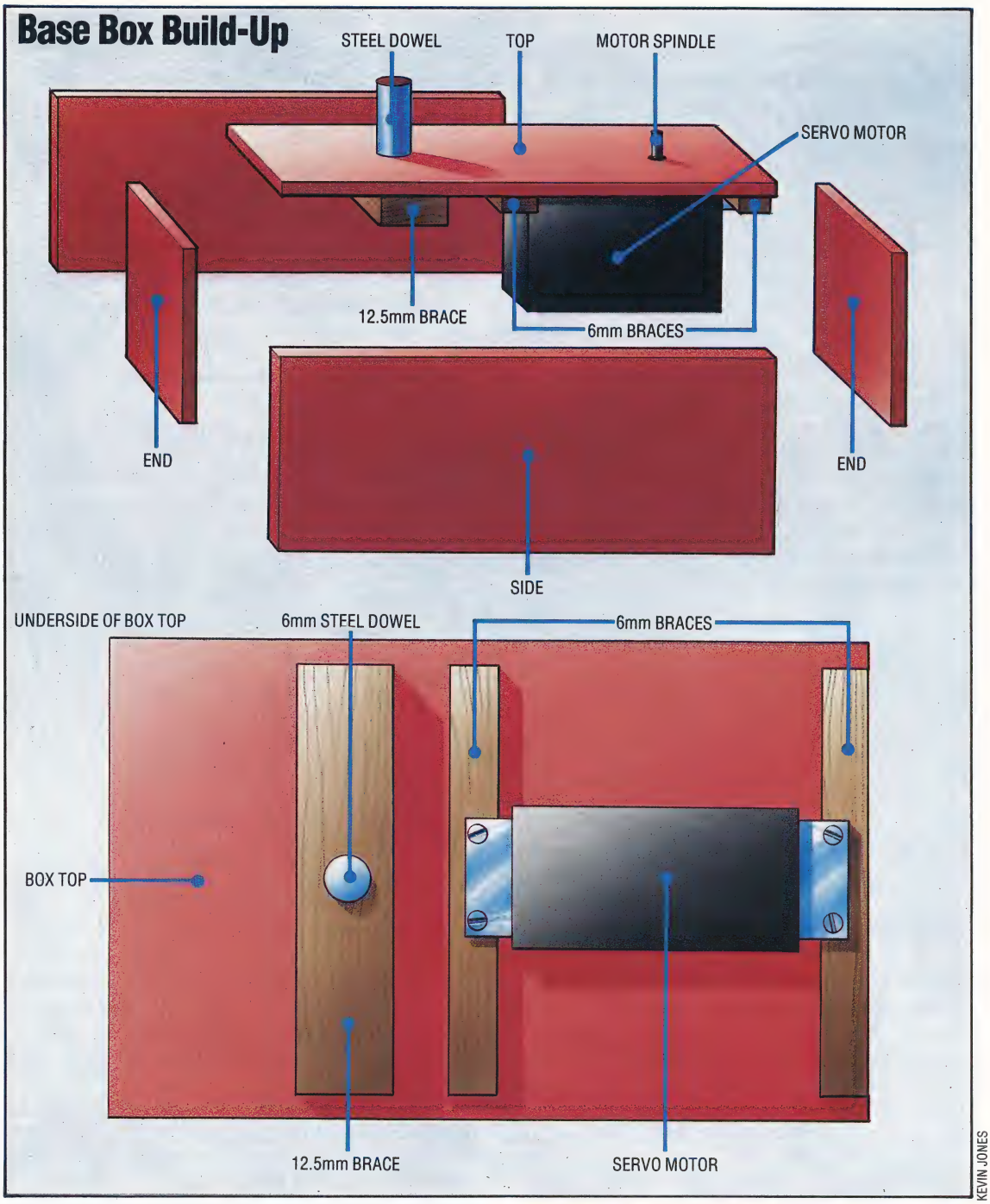
The main arm assembly is attached to the base

Parts List

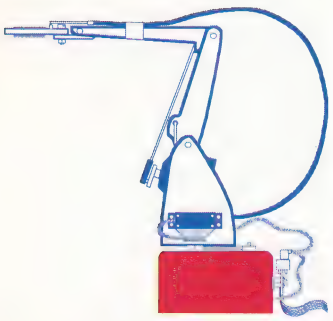
- A number of 8 BA machine screws and nuts to assemble the claw
- 5mm, 6mm internal diameter spacer
- Wood and epoxy glues
- 16 10mm self-tapping screws and rubber grommets to mount the motors
- 5cm length of foam for the claw jaws
- Solder and insulating tape

Pattern Cutouts
Make tracings of the patterns for the frame parts, stick them onto the plywood board and cut them out. Note that some patterns are used twice to produce pairs of parts. Match arm pairs and put them together before drilling out to full size later. Sand down each part carefully to remove rough edges

Base Box Build-Up



KEVIN JONES



box, by way of a ball-race pivot. The ball race should be flanged and have an internal diameter of 6mm. Such ball races are often used for the front wheels of model cars and should be obtainable from modellers' shops. To lengthen the bearing, we also require a 25mm length of 6mm steel dowel. This dowel will be firmly affixed to the base and fit snuggly through the ball-race.

The claw is operated from a motor mounted lower down on the arm assembly. The motor should be connected to the claw using a 500mm long flexible link (sometimes called a 'snake'). This is essentially a steel wire enclosed in a plastic sleeve. Model aircraft often use this type of linkage to operate flaps. A 50mm length of stiff wire (piano wire is ideal) will be needed to open and shut the claw, and this must be bound to the end of the snake with 5A fuse wire. The arm members are moved using a system of push-rods. Four 150mm lengths of 2mm steel rod should be used for this. The rods can be connected to the motor discs, either directly, or with small ball-joints.

Constructing The Base Box

Start by gluing the 12.5mm brace onto the top of base box. The main pivot hole should then be drilled out to take the 6mm steel dowel. Drill right through the box top and brace, insert the dowel and glue it into position. Glue the other two brace pieces in position on the box top. These braces are used to mount one of the servo motors. Drill out the other hole in the box top so that the motor spindle can fit through, and screw the motor in place onto the braces using the four self-tapping screws and rubber grommets. The box sides and ends can then be glued into position. When dry, sand off any rough edges and corners

DATA ORGANISATION

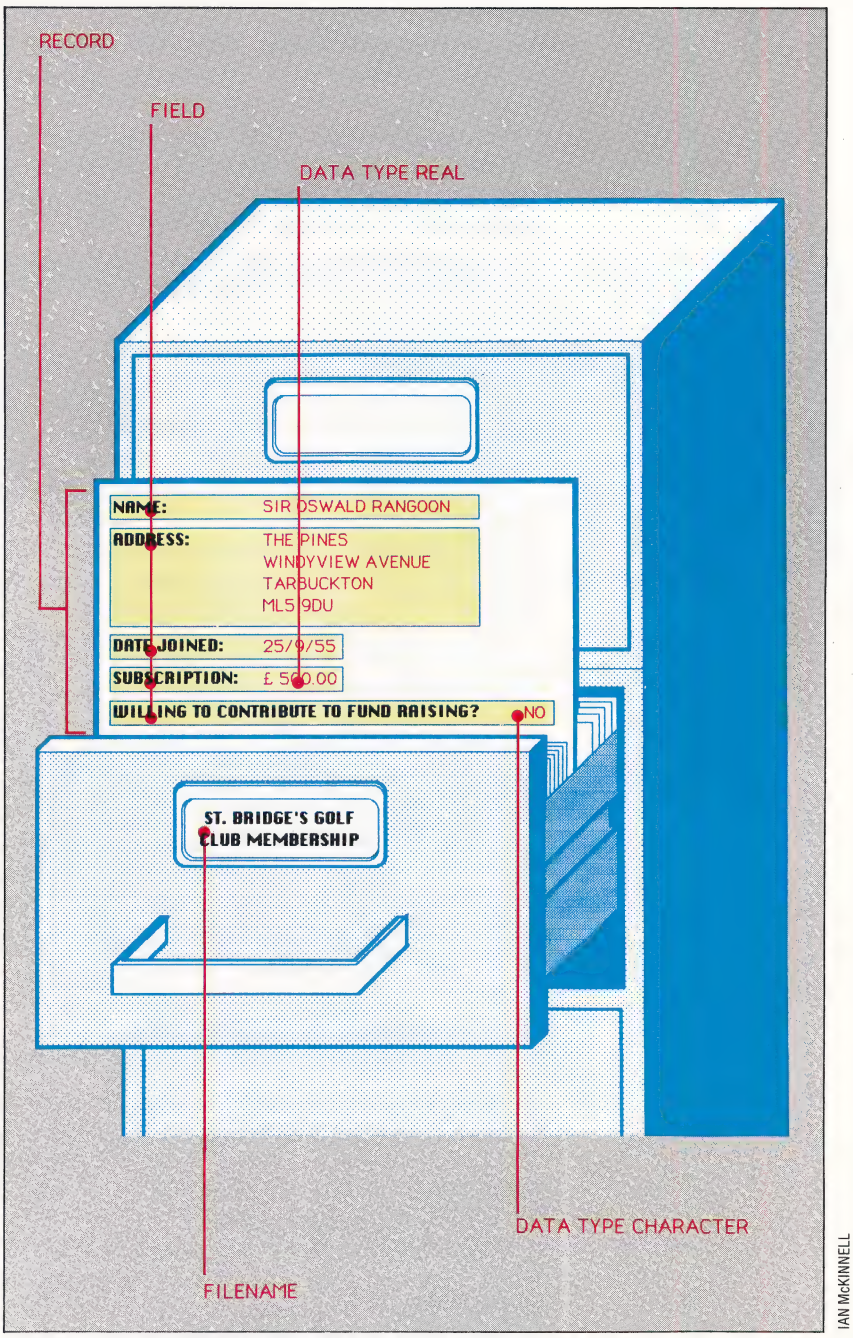
A properly constructed database will organise information, whether large or small amounts, and allow the quick and easy access to any one item or combination of data. In this initial instalment of our series on databases, we illustrate the thinking behind setting up an efficient and well structured system.

People who have to manipulate large bodies of structured data range from club secretaries to newsagents, estate agents to hospital administrative staff. A computer and a database manager program can help keep data organised and up to date. But before it is practical to talk about how database programs are used, it is important to review some of the fundamental concepts involved.

To the computer programmer, data can be viewed as being either structured or unstructured. Unstructured data are isolated units, usually stored in variables or constants. Examples might include NUMBER_OF_GOES:= NUMBER_OF_GOES +1 or IF HIT = 1 THEN PROCEDURE INCREMENT. In these two examples, NUMBER_OF_GOES and HIT are variables; in the first case, the value of the variable is being changed, and in the second, it is being tested (with a possible branch to a procedure). It is clear that the objects NUMBER_OF_GOES and HIT can have only one value at a time.

Computer programmers, especially those familiar only with BASIC, are accustomed to thinking of data as being unstructured in this way, but in real life we are used to thinking of structured data and take such structures for granted. Think of all your friends, for example. You certainly have your own 'data structure' in this respect. You know their names, you know their sex, you know their age (approximately), you probably know their jobs, and you may know their salaries and other things as well. We call a related set of facts such as this (about one person, concept or object) a record. Each record consists of one or more fields (which may be of similar or dissimilar data types) and the whole set of records is called a file.

To put this into context, consider a golf club secretary who has to administer 500 members. To keep things in order, he maintains a card index — one card for each member. Each card contains various sorts (types) of information (data) about a single member. The information might include NAME, FORENAME, SEX, AGE, YEAR JOINED, SUBSCRIPTION PAID?, SALARY, MEMBER NUMBER, and so on. In computer terminology, each of these pieces of data within the record is known as a field, and the various fields contain different types of data. NAME and FORENAME will obviously be character strings, so 1374662 would not be an appropriate piece of data for a member's name. SEX input will be a Boolean type, capable of taking only one of two binary values. AGE will be an integer type (assuming that we do not need to know that a member is 37.624 years old.). YEAR JOINED will be an integer value with a restricted range. We'll assume that YEAR*



IAN McKINNELL

JOINED = 1066 is inappropriate, even if the file contains records of past members Also, YEAR JOINED = 2001 is unlikely unless the club rules allow children to put their names down for future membership. SALARY would be data of type real; someone might earn a salary of £12,345.67.

Programmers need to be keenly aware of the data types available in the chosen programming language. Some languages, including BASIC and C, are weakly typed, while others such as PASCAL insist on all data items conforming to strictly defined types.

If it's not clear how this is relevant, let's first look at how BASIC and PASCAL deal with data before we define a database. BASIC recognises only two data structures — variables and files. If further structuring of data is required in a BASIC program, the programmer will have to impose it because the language does not provide it. BASIC variables may be of a number of different types, including integer values (SCORE% = 7 for example), single precision reals (SALARY! = 1234.56), double precision reals (GARETH'S SALARY = 123456.66666666666) and string variables (WONDERFUL$ = 'The Home Computer Advanced Course'). There is no such thing in BASIC as a constant type, such as PI = 3.141592 — of course, you can always set that value to the variable PI, but you can just as easily include the statement PI = 6.2 later in the program.

Other programming languages — PASCAL is probably the best known example — provide several pre-defined types and allow programmers to define other, new ones in terms of the existing types. The pre-defined types in PASCAL are constants (non-variables) and variables of type char (characters), integer (whole numbers), real (real numbers such as 12.71) and boolean (binary data that is either true or false). Sets, arrays, records and files are built in, and new data types can be easily defined by the programmer. Here's how you would create the data type DAYS.

```
TYPE
  DAY=(MONDAY,TUESDAY,WEDNESDAY,
       THURSDAY, FRIDAY, SATURDAY, SUNDAY);
```

Any variable used in the program of type DAY could only have one of the values specified, so if DAY_OFF was defined as being of type DAY, DAY_OFF :=NOVEMBER would be illegal, but DAY_OFF := SUNDAY would be acceptable.

From the computer programmer's point of view, there are pros and cons to the subject of strong typing; sometimes it gets in the way, sometimes it helps prevent programming errors. We have covered a few of the concepts of defining types of data, so now let's look at the relevance of data structures to databases.

WHAT IS A DATABASE?

A database is any collection of structurally related data. Conventionally, the data will be organised as a file (with its own filename), the file will consist of records (each with its own record number), and the records will consist of one or more fields of data, the fields containing one or more data types.

Supposing your database is organised as a card index in a box, the primary key to each entry will almost certainly be a name, arranged alphabetically. If there is a need to look for any member called Svensen, you will flick through the cards until you find the S file, then slow down until you find any names beginning with Sv until you finally locate any entries for Svensen. If, however, you were asked to find how many female members you have, finding the answer would be more tedious. The task of finding the data would be even more time consuming if someone wanted to know how many members are over 36 years old and earn £11,000 a year.

But this is exactly the sort of thing a database program can do at the touch of a few computer keys. Strictly speaking, the database is the information, and the program that helps you enter and manipulate it should be called a 'database manager'. To maintain this distinction between the collection of data and the program that manipulates it, we'll stick to the convention of calling the program a database manager, or DBM for short.

DATABASE MANAGERS

DBMs vary in their capabilities considerably. The simplest ones are little more than electronic address books able to retrieve a record when given a simple parameter such as NAME = ? The more sophisticated DBMs have their own built-in programming languages that allow the information in the database to be manipulated in highly complex ways, including exporting data fields to other programs (such as payrolls, invoicing software or word processors). In this series, we will look at DBMs, from the cheap and cheerful to the very advanced, and learn how to use them properly. Before we do that, however, let's review the preamble about data typing. A good DBM will allow the user to define the type of data allowable for each field, and will reject an entry such as NAME = 143326 or YEAR JOINED = 1066. It would also report an error if you tried to find records conforming to IF NOT (MALE OR FEMALE) AND SALARY<0. Good DBMs, unlike programming languages, need to have strong type checking to prevent erroneous data from being input when the records are created or modified.

In further instalments of this series, we'll look at how various DBMs organise their records. We'll consider relational and multi-file databases, fixed length versus variable length fields and records, creating databases and extracting data from databases. To illustrate the concepts involved, we'll concentrate on three DBMs — Psion's Archive, Ashton Tate's dBase II and Caxton Software's Card Box, with quick glances at various other packages. Programming examples will use the syntax of the products considered, though general examples will be written in BASCAL, a pseudo-language hybridised from BASIC and PASCAL.

THE MAIN EVENTS

Unlike interrupts (which we examined in the previous instalment, see page 1077), 'events' are a category of interrupt signals that the CPU can be made to ignore. We look at some of the more useful events available on the BBC Micro.

There is a wide variety of situations in which 'event' signals are generated in the BBC Micro: such as when characters are entered into the input buffer, after the generation of a vertical sync signal, on completion of a conversion on the A/D converter and when the Escape key is pressed. Some of these are machine occurences that also generate interrupts, which are dealt with through the IRQV1 vector (see page 1078).

Events thus enable us to make use of some of these high priority IRQs by a rather 'back door' method. In order for events to be generated and acted upon by the computer they have to be *enabled*, or 'turned on'. Once a particular event is enabled, whenever it occurs the CPU jumps to a routine whose start address is held in the vector at addresses &220 and &221. To stop this routine being executed when an event is generated, we have to *disable* the event. Enabling and disabling events is done by using two OSBYTE calls. The following table shows the events available and the OSBYTE calls used to enable or disable them. We have included only the most important and commonly used events.

Commands To Enable And Disable Some Useful Events

Event	Enable	Disable
Output buffer empties	*FX14,0	*FX13,0
Input buffer fills up	*FX14,1	*FX13,1
Character enters input buffer	*FX14,2	*FX13,2
A/D conversion completed	*FX14,3	*FX13,3
Start of vertical sync	*FX14,4	*FX13,4
Interval timer has 'timed out'	*FX14,5	*FX13,5
Escape has been pressed	*FX14,6	*FX13,6

Thus, to enable the Interval timer has 'timed out' event, we'd simply execute *FX14,5 from BASIC, or its equivalent from machine code. After this event has been enabled, the routine which has its address held in the event vector will be executed whenever the interval timer reaches zero. For this reason, it's vital that a piece of machine code should be at that address, otherwise a crash is likely to occur.

You will have no doubt noticed that there is one event vector and yet a range of different events. Clearly, all the enabled events cause a jump to the same routine, which must therefore be written to take all the enabled events into account. How do we know what event caused the event handling routine to be entered? If we've only enabled one event, then the answer is obvious; otherwise, we have to get the answer from the contents of the A register when the routine is entered. The following table shows the contents of various registers on entry to the routine pointed to by the event vector.

Contents Of Registers On Entry To Event Routines

Event	A Register	X Register	Y Register
Output buffer	0	Buffer number	#
Emptying input buffer	1	Buffer number	#
Emptying character in input buffer	2	#	Character
A/D converter	3	#	Channel number
Vertical sync	4	#	#
Interval timer	5	#	#
Escape condition	6	#	#

In this table, # indicates that the register contents are not specified. This means that it's possible, on entering the event handling routine, to check the value in the A register so that we can see what event caused the routine to be entered.

When you write an event handler routine, you must save the register as the first task on entering the routine. Unlike the BBC Micro's interrupt handling facilities, the event handler does *not* automatically save the contents of the A register and the status register to the stack, and so we must do this job. The event handler returns control to the interrupted program with an RTS instruction. Enough of the theory; let's examine the events in turn and consider several examples.

- *Event 0:* This is generated when the output buffer empties. The buffer number that is used is the same as that used by OSBYTE 21, which we looked at when we discussed the OSBYTE call in detail (see page 957).
- *Event 1:* Event 1 is generated when the current input buffer fills up and an attempt is made to put another character into it. The number specified is the same as that for OSBYTE 21. The ASCII code in the Y register on entry to the event handler is the code that could not be put into the buffer.
- *Event 2:* This is generated whenever a character enters the input buffer — usually after a key is pressed. The program that follows demonstrates the basic elements of an event handler, generating a VDU 7 whenever a key is pressed. This will happen even if the keys are pressed when a BASIC program is running.

```
10 DIM mc% (100)
20 FOR I%=0 TO 2 STEP 2
```

```
30 P%=mc%
40 [ OPT I%
50 PHP
60 PHA
70 TXA:PHA
80 TYA:PHA
90
100 LDA #7
110 JSR &FFE3
120
130 PLA:TAY
140 PLA:TAX
150 PLA
160 PLP
170 RTS
180
190 ] :NEXT I%
200 ?&220=mc% MOD 256:REM low byte of
    address placed in vector
210 ?&221=mc% DIV 256:REM high byte of address
220 *FX 14,2
230 REM only enable the event after we've set the
240 REM routine up
250 REPEAT
260 PRINT I%
270 I%=I%+1
280 UNTIL FALSE
```

● *Event 3:* Event 3 is generated whenever one of the four channels of the BBC Micro's A/D converter has completed a conversion. This event will happen every five or 10 milliseconds, depending upon the conversion rate that has been selected for the converter.

● *Event 4:* This event is generated at the start of the vertical sync pulse. It occurs 50 times a second, and can be of use as a source of timing pulses.

● *Event 5:* Probably the most useful of the events available on the BBC Micro. As well as the TIME variable, the BBC Micro's operating system updates an *interval timer* every one hundredth of a second. We have already discussed this timer with regard to its use by certain OSWORD calls (see page 978). The timer is regularly incremented and when it reaches zero an event is generated. It is clear that this enables us to interrupt normal processing at regular intervals to perform a particular job. The interval timer is written to using the OSWORD with A=4 call and is read from using OSWORD with A=3.

If we wanted to get the system to generate an event after one second, we would have to load the interval timer with a value of 100 less than the maximum value that can be represented by it. As the timer is incremented, it would then cross zero after 100 'ticks'. Because it is a five byte timer, its maximum value before it crosses zero is &FFFFFFFFFF. This program shows a demonstration of this event, generating a regular 'beep':

```
10 DIM mc% (100), delay% 10
20 FOR I%=0 TO 2 STEP 2
30 P%=mc%
40 [ OPT I%
50 PHP:PHA
60 TXA:PHA
70 TYA:PHA
80 LDA #7:JSR &FFE3
90 .clock set  / reset the clock to give next interrupt
100 LDX #delay% MOD 256 / set the OSWORD
    block up
110 LDY #delay% DIV 256
120 LDA #4
130 JSR &FFF1
140 PLA:TAY
150 PLA:TAX
160 PLA:PLP
170 RTS
180 ]:NEXT I%
190 ?&220=mc% MOD 256
200 ?&221=mc% DIV 256
210 !delay%=&FFFFFF9C:delay%?4=&FF
220 *FX 14,5
230 CALL clock set:REM set the clock running
240 END
```

● *Event 6:* This is generated by pressing the Escape key.

There are three other events available to us, and these signify an error on the RS423 system, an Econet error or a User Specified Event. That completes our survey of the most useful events used by the BBC Micro's OS. In the next instalment we will give a concluding overview of the operating system in action.

Exercise Your Knowledge

The following program demonstrates some of the principles discussed so far in our series on the BBC Micro's OS. Examine the listing carefully, then enter the program and RUN it. Try to find out (1) what operation is performed by the program and (2) how the program works. A full explanation, together with other sample programs, will appear in the next instalment of the series. Hint: the variable name in line 80 should put you on the right track

```
 20 PROCassemble
 30 END
 50 DEFPROCassemble
 70 DIM MC% &FF
 80 oldvec=?&20E+256*?&20F
 90 FOR pass=0 TO 3 STEP 3
100 P%=MC%
120 [OPT pass
130.temp EQUB 00
140.start
150 PHP
160 STA temp
170 TXA:PHA
180 TYA:PHA
200 JSR check
230 PLA:TAY
240 PLA:TAX
250 LDA temp
260 PLP
270 JMP oldvec
290 .check
300 LDA temp
310 CMP#65
320 BMI out
330 CMP #91
340 BPL out
350 ADC #32
360 STA temp
370 .out RTS
390 ]:NEXT
400 ?&20E=start MOD 256
410 ?&20F=start DIV 256
420 ENDPROC
```

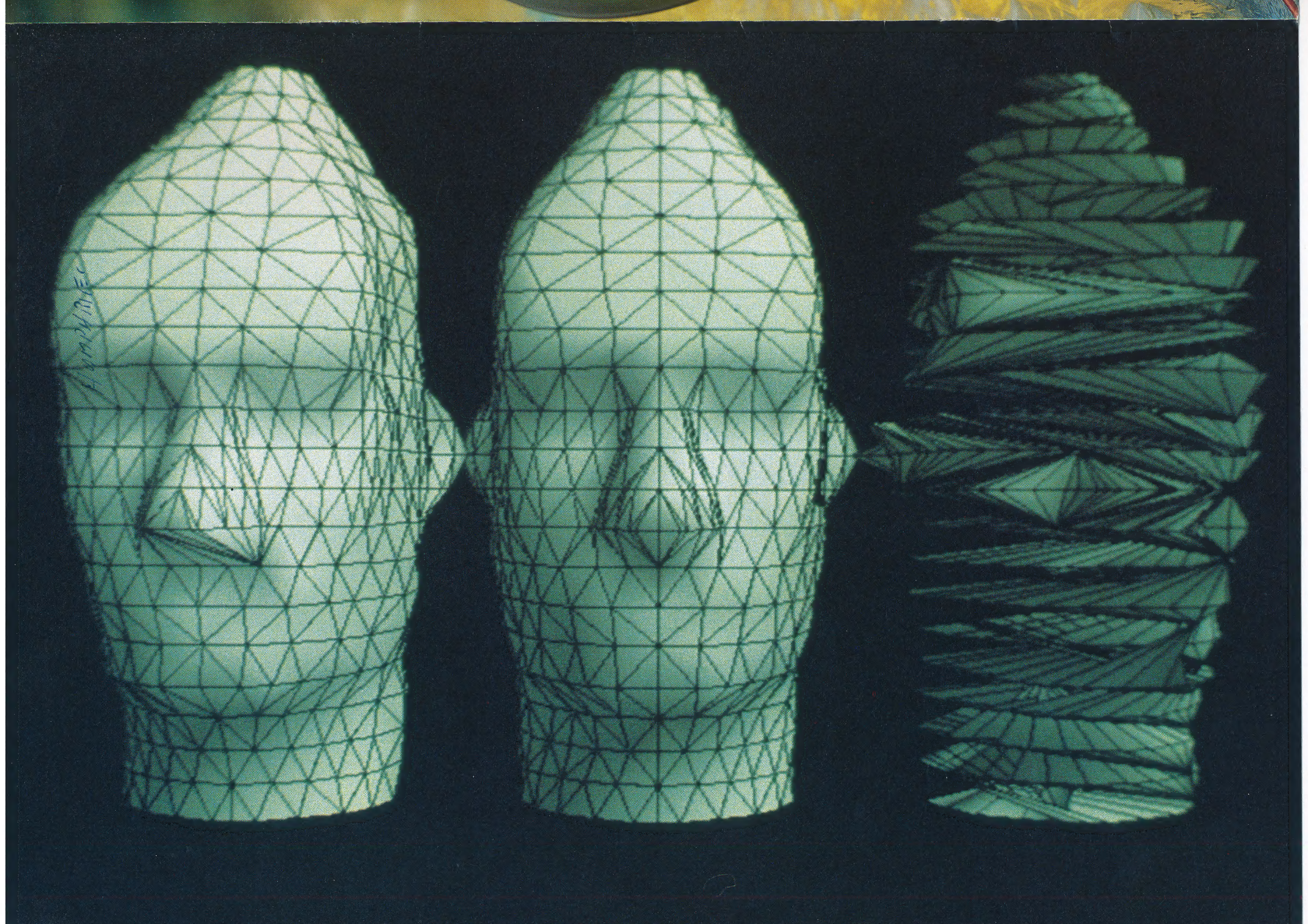
DATABASE

Here, courtesy of Acorn Computers, we publish the first instalment in a series listing the call addresses of the BBC Micro's BASIC ROM routines. Note that these addresses are for BASIC II only

ALPHABETICAL ROM INDEX

This is an alphabetical listing of the routines in the ROM:

! binary operator	967F
! unary operator	95A5
$ unary operator	95B0
* operator routine	9D3C
* statement routine	8B73
+ operator routine	9C4E
- operator routine	9CB5
- unary operator	AD8C
/ operator routine	9DE5
< operator routine	9BC0
< = operator routine	9BD4
<> operator routine	9BDF
= operator routine	9BAE
= statement routine	8B47
> operator routine	9BE8
> = operator routine	9BFA
? binary operator	967B
? unary operator	95A7
ABS function routine	AD6A
ACS function routine	A8D4
ADVAL function routine	AB33
AND operator routine	9B7A
ASC function routine	AC9E
ASN function routine	A8DA
ATN function routine	A907
AUTO command routine	90AC
Assemble a single instruction	85BA
Assembler base op-codes	84C5
Assembler entry	8504
Assembler exit	84FD
Assembler label definition	85A5
Assembler mnemonics table (LSB)	8451
Assembler mnemonics table (MSB)	848B
Assign value to numeric variable	B4B1
BASIC error handler	B433
BGET statement routine	BF6F
BPUT statement routine	BF58
BRK handling routine	B402
CALL statement routine	8ED2
CHAIN statement routine	BF2A
CHR$ function routine	B3BD
CLEAR statement routine	928D
CLG statement routine	8EBD
CLOSE statement routine	BF99
CLS statement routine	8EC4
COLOUR statement routine	938E
COS function routine	A98D
COUNT function routine	AEF7
Check FAC#1	A1DA
Check for a 'Bad program'	BE6F
Check for a comma at PTR#2	8AAE
Check for end of statement	9857
Check if A is alphanumeric	8926
Clear a new variable	9531
Clear stacks	BD3A
Clear variables	BD20
Cold start	8ADD
Compare two items	9A9E
Convert a line number to binary	889C
Convert number to a string	9EDF
Convert string to a number	A07B
Create a PROC/FN catalogue entry	94ED
Create a catalogue entry	94FC
DATA statement routine	8B7D
DEF statement routine	8B7D
DEG function routine	ABC2
DELETE command routine	8F31
DIM statement routine	912F
DIV operator routine	9E0A
DRAW statement routine	93E8
Decode hexadecimal number	AE6D
Decrement MAN#1	A4C7
Delete a line from program	BC2D
Delete bytes in buffer	887C
Discard the IAC from stack	BDFF
END statement routine	8AC8
ENDPROC statement routine	9356
ENVELOPE statement routine	B472
EOF function routine	ACB8
EOR operator routine	9B55
EQUX in assembler	883A
ERL function routine	AF9F
ERR function routine	AFA6
EVAL function routine	ABE9
EXP function routine	AA91
EXT function routine	BF46
Error handler	B402



© 1983 LEWIS, J.—MIT